

BACHELORARBEIT

Evolution neuronaler Netze in einer physikalischen 2D-Umgebung

von

Dominik Colling

abgegeben am 01.09.2011 beim

Institut für Angewandte Informatik und
Formale Beschreibungsverfahren (AIFB)
des Karlsruher Instituts für Technologie

Referent: Prof. Dr. Hartmut Schmeck

Betreuer: Lukas König

Betreuer: Daniel Pathmaperuma

Anschrift:

Rheinbergstraße 47

76187 Karlsruhe

Inhaltsverzeichnis

Abstract	xi
1 Einleitung	1
1.1 Motivation	1
1.2 Geschichte der Evolutionstheorie	1
1.3 Künstliche Evolution in der Informatik	3
1.4 Gliederung	4
2 Grundlagen	5
2.1 Evolutionäre Robotik	5
2.2 Genetische Algorithmen	6
2.2.1 Selektion	7
2.2.2 Rekombination	8
2.2.3 Mutation	10
2.3 Künstliche Neuronale Netze	10
2.3.1 Neuro-Evolution	13
2.4 Initial Training und Lifelong Adaptation	15
2.4.1 Training Phase Evolution	15
2.4.2 Lifelong Adaption By Evolution	16
2.5 Pseudozufallszahlen	18

3 Verwandte Arbeiten	19
3.1 Dario Floreano und Francesco Mondada	19
3.1.1 Versuchsaufbau	20
3.1.2 Ergebnis und Analyse	22
3.2 Karl Sims	22
3.2.1 Ergebnisse	24
3.3 Weitere Arbeiten	25
3.3.1 Stefano Nolfi und Domenico Parisi	25
3.3.2 T. Köpsel, A. Noglik und J. Pauli	27
4 Implementierung	29
4.1 Simulationsumgebung	29
4.2 Umwelt	30
4.3 Agenten	30
4.4 Gehirn	32
4.5 Neuronales Netz	33
5 Experimente	35
5.1 Versuchsaufbau	35
5.2 Speicherung	36
5.3 Verwendete Parameter	37
6 Auswertung	39
6.1 Erste Versuchsreihe - Start vom Planeten und aus dem Orbit	39
6.1.1 Start vom Planeten	39
6.1.2 Start aus dem Orbit	47
6.1.3 Fazit	54
6.2 Zweite Versuchsreihe - Start aus dem Orbit mit einer Drehung der Raketen um 90 Grad	54
6.2.1 Fazit	58

6.3	Dritte Versuchsreihe - Start aus dem Orbit mit Optimierung der Topologie	60
6.3.1	Start mit normaler Ausrichtung	60
6.3.2	Start mit einer Drehung um 90 Grad	61
6.3.3	Fazit	66
7	Zusammenfassung und Ausblick	67
7.1	Zusammenfassung	67
7.2	Ausblick	68
A	Weitere neuronale Netze	69
A.1	Start aus dem Orbit aus der Höhe 100 LE	69
A.2	Start aus dem Orbit mit einer Drehung um 90 Grad	69

Abbildungsverzeichnis

2.1	Beispiele für Crossover	9
2.2	Beispiele für eine Mutation	10
2.3	Aktivierungsfunktionen von Neuronen	11
2.4	Beispiel für ein neuronales Netz	12
2.5	Verfahren für Training Phase Evolution und Lifelong Adaption by Evolution	17
3.1	Spielfeld beim Versuch von Floreano und Mondada	20
3.2	Fitnesswerte beim Versuch von Floreano und Mondada	23
3.3	Trajektorien beim Versuch von Floreano und Mondada	23
3.4	Ergebnisse von Karl Sims	26
4.1	Umwelt für die Simulation	31
4.2	Koordinatensystem aus Sicht der Rakete	32
4.3	Grafische Darstellung des verwendeten neuronalen Netzes	34
5.1	Ablauf der Speicherung der Ergebnisse	37
6.1	Flugbahn einer Rakete Kategorie B1	40
6.2	Neuronales Netz einer Rakete der Kategorie B1	41
6.3	Flugbahn einer Rakete der Kategorie B2	42
6.4	Neuronales Netz einer Rakete der Kategorie B2	42
6.5	Flugbahn einer Rakete der Kategorie B3	43

6.6	Neuronales Netz einer Rakete der Kategorie B3	44
6.7	Start der Rakete der Kategorie B4	45
6.8	Flugbahn der Rakete der Kategorie B4	45
6.9	Neuronales Netz der Rakete der Kategorie B4	46
6.10	Flugbahn einer Rakete der Kategorie L1	48
6.11	Neuronales Netz einer Rakete der Kategorie L1	49
6.12	Flugbahn einer Rakete der Kategorie L2	50
6.13	Neuronales Netz einer Rakete der Kategorie L2	50
6.14	Flugbahn der Rakete der Kategorie L3	52
6.15	Neuronales Netz der Rakete der Kategorie L3	53
6.16	Flugbahn einer Rakete der Kategorie L4	56
6.17	Flugbahn einer Rakete der Kategorie L5	56
6.18	Flugbahn einer Rakete der Kategorie L6	57
6.19	Flugbahn einer Rakete der Kategorie L7	58
6.20	Flugbahn der Rakete der Kategorie L8	59
6.21	Neuronales Netz einer Rakete der Kategorie L1 mit verstecktem Neuron	62
6.22	Flugbahn einer Rakete der Kategorie L4	63
6.23	Flugbahn einer Rakete der Kategorie L9	64
A.1	Flugbahn einer Rakete der Kategorie L5	69
A.2	Weiteres neuronales Netz der Rakete der Kategorie L3	70
A.3	Neuronales Netz einer Rakete der Kategorie L4	70
A.4	Neuronales Netz der Rakete der Kategorie L5	71
A.5	Neuronales Netz einer Rakete der Kategorie L6	71
A.6	Neuronales Netz einer Rakete der Kategorie L7	72
A.7	Neuronales Netz einer Rakete der Kategorie L8	72

Tabellenverzeichnis

5.1	Übersicht über alle in den Versuchen verwendeten Parameter Teil 1	37
5.2	Übersicht über alle in den Versuchen verwendeten Parameter Teil 2	38
6.1	Ergebnisse der Versuche beim Start vom Planeten	44
6.2	Ergebnisse der Versuche beim Start aus der Höhe 50 LE	49
6.3	Ergebnisse der Versuche beim Start aus der Höhe 100 LE	51
6.4	Ergebnisse der Versuche beim Start aus der Höhe 150 LE	53
6.5	Ergebnisse der Versuche beim Start aus der Höhe 100 LE und einer Drehung der Agenten um 90 Grad	59
6.6	Ergebnisse der Versuche beim Start aus der Höhe 100 LE mit Optimierung der Topologie	60
6.7	Ergebnisse der Versuche beim Start aus der Höhe 100 LE und einer Drehung der Agenten um 90 Grad mit Optimierung der Topologie	65

Abstract

In dieser Arbeit wird die Evolution von Steuerprogrammen für Agenten in einer simulierten physikalischen Umgebung untersucht. Dabei wird als Beispielszenario das Umfliegen eines Planeten durch Raketen-Agenten in einem durch Minimal- und Maximalentfernung eingeschränkten Orbit betrachtet. Das Szenario wird im Java-Framework Easy Agent Simulation (EAS) implementiert.

Die Agenten sind mit elementaren Eigenschaften einfacher Raketen ausgestattet. Sie besitzen einen Antrieb sowie die Möglichkeit sich zu drehen. In der simulierten physikalischen 2D-Umgebung befindet sich ein Planet, der eine Anziehungskraft auf die Agenten ausübt. Die Agenten sollen in eine stabile Umlaufbahn gelangen. Sie werden dabei von einem künstlichen neuronalen Netz (KNN) gesteuert, das sich im Laufe der Simulation entwickeln soll, bis die Agenten nicht mehr abstürzen bzw. sich zu weit vom Planeten entfernen. Um dieses Verhalten zu erreichen, wird jedes KNN mithilfe von Genetischen Operatoren evolviert. Es werden die Synapsen im KNN sowie teilweise zusätzlich die Struktur des KNN als Ganzes optimiert. Dabei erhalten die KNN von Agenten mit längerer Lebensdauer einen höheren Fitnesswert und damit eine höhere Wahrscheinlichkeit in der nächsten Generation wiederverwendet zu werden.

Es konnte gezeigt werden, dass die Evolution sowohl bei einem Start vom Planeten als auch bei einem Start aus dem Orbit erfolgreich verlaufen kann und sich in beiden Fällen KNN entwickelten, die es dem Agenten ermöglichten, während der Simulation nicht abzustürzen. Zusätzlich wurde bewiesen, dass die Startrichtung sowohl Einfluss auf die entwickelten Verhalten als auch auf die erreichten Fitnesswerte hat. Eine zusätzliche Optimierung der Struktur der KNN dagegen verändert die Verhaltensweisen der Agenten nicht.

Kapitel 1

Einleitung

1.1 Motivation

In dieser Arbeit wird in einer Simulation untersucht, ob sich mithilfe von naturinspirierten Lernverfahren Kontrollsysteme für Raketen entwickeln können. Dafür werden Agenten mit elementaren Eigenschaften von Raketen, d.h. sie besitzen einen Antrieb und können sich drehen, in einer Umwelt, in der die Gesetze der Physik gelten, getestet. Die Umwelt besteht aus einem Planeten, der eine Gravitationsquelle darstellt, und dem All um ihn herum. Die Agenten werden von neuronalen Netzen gesteuert, die sich im Laufe der Simulation entwickeln sollen. Diese sollen die Raketen so steuern, dass sie nicht abstürzen und stattdessen in eine stabile Umlaufbahn gelangen. Das hier verwendete naturinspirierte Verfahren kann der Evolutionären Robotik zugeordnet werden. Dabei wird auf Mechanismen zurückgegriffen, die in der natürlichen Evolution wirken und dazu geführt haben, dass sich die Artenvielfalt der Lebewesen entwickelt hat.

1.2 Geschichte der Evolutionstheorie

Die Geschichte der Evolutionstheorie beginnt am Anfang des 19. Jahrhunderts. Damals verfasste Jean-Baptiste de Lamarck sein Werk „Philosophie Zoologique“ [Lam09]. Er war einer

der Ersten, der die Theorie anzweifelte, dass alle Arten, so wie sie waren, von Gott geschaffen wurden und sich nicht mehr veränderten. Lamarck war der Meinung, dass sich die Lebewesen im Laufe der Zeit weiterentwickeln, ihre Struktur immer komplexer wird und sie diese an ihre Nachfahren vererben. So verbessern sich Organe, die oft genutzt werden, während die ungenutzten langsam verkümmern. Als Beispiel nannte er die Giraffe, deren Hals sich dadurch, dass sich Blätter nur hoch in den Bäumen befinden, immer länger würde.

1859 veröffentlichte Charles Darwin sein Werk „On the Origin of Species“ [Dar59]. Darin ging er noch einen Schritt weiter. So ging er davon aus, dass alle Lebewesen eine gemeinsame Abstammung haben und sich die verschiedenen Arten erst im Laufe der Evolution entwickeln. Als wichtigsten Antrieb dafür nannte er die natürliche Selektion. Nach Darwin gibt es innerhalb jeder Art eine natürliche Variabilität. Da es mehr Nachkommen als Kapazität im Lebensraum gibt, herrscht Konkurrenz. In dieser Konkurrenz setzen sich dann die Individuen, die besser angepasst sind, deswegen durch, weil sie eine höhere Überlebenswahrscheinlichkeit haben und sie damit auch mehr Nachkommen zeugen können, die ihre Merkmale tragen und weitervererben. Sechs Jahre später erklärte Gregor Mendel in seinem Werk „Versuche über Pflanzenhybriden“ [Men66] mit den Mendelschen Regeln an einem praktischen Beispiel wie die Evolution funktioniert. Er kreuzte Erbsenpflanzen und beschrieb damit, wie die Vererbung von Merkmalen funktioniert, die von nur einem Gen abhängen.

Nach heutiger Erkenntnis ruht die Evolution auf vier Säulen [FM08]:

- Population: Um eine Evolution zu erreichen, muss es mindestens zwei Individuen geben. Gibt es nur ein Individuum kann man nicht von Evolution sprechen.
- Diversität: Diversität bedeutet, dass die Individuen sich in einem gewissen Ausmaß unterscheiden müssen.
- Vererbung: Eigenschaften können an Nachfahren durch Reproduktion vererbt werden.
- Selektion: Nur ein Teil der Population vererbt seine Eigenschaften und zwar derjenige, der an die natürlichen Bedingungen am besten angepasst ist. Diese Individuen besitzen die höchste Fitness.

Es müssen nicht immer die individuell Besten diejenigen sein, die im Vorteil sind. Es kann auch vorkommen, dass durch altruistisches Verhalten diejenigen im Vorteil sind, die in einer Gruppe zusammenarbeiten.

Wichtig ist auch festzuhalten, dass die Fitness immer von der aktuellen Umwelt abhängig ist. Sie ist also immer von Ort und Zeit bestimmt. Eine Änderung der Umwelt kann auch zu einer Änderung der Auswahlkriterien führen, nach denen selektiert wird.

Die Diversität entsteht während der Reproduktion durch fehlerhaftes Kopieren der Erbinformation. Die sogenannten Mutationen. Dadurch können neue oder veränderte Eigenschaften entstehen. Verbessern sich die Eigenschaften eines Lebewesen, hat dieses eine höhere Fitness und die Chance, dass es die neuen Eigenschaften weitervererbt, steigt. Gibt es zumindest keine Verschlechterung und werden die neuen Eigenschaften weitervererbt, spricht man von neutraler Evolution. Diese kann dann in folgenden Generationen durch weitere Veränderung doch noch Auswirkungen zeigen.

1.3 Künstliche Evolution in der Informatik

Mithilfe dieser Erkenntnisse wurden vor mehr als 50 Jahren die ersten naturinspirierten Algorithmen zum Finden von Lösungen in der Informatik und im Ingenieurbereich entwickelt. Diese greifen unter anderem auf die oben beschriebenen Theorien zurück, um Optimierungsaufgabe approximativ zu lösen bzw. um bereits gefundene Lösungen zu verbessern. Dabei wird ein abstraktes Objekt, das eine Lösung für ein Problem darstellt, wie ein Organismus behandelt. Dieses wird dann mithilfe von Evolutionären Operatoren, die meist mit Zufallszahlen arbeiten, verändert, reproduziert und bewertet (vgl. Kapitel 2.2 und [Wei07]).

Der Unterschied zwischen der künstlichen Evolution, also vom Menschen genutzten Algorithmen, und der natürlichen Evolution ist, dass in der künstlichen Evolution immer eine Lösung für ein festes Ziel optimiert werden soll, während die Evolution in der Natur ein offenes Ende hat. In der Natur zeichnet sich ein Individuum mit hoher Fitness dadurch aus, dass es viel Nachwuchs zeugt, in der künstlichen Evolution durch eine explizite Güte der gefundenen Lösung.

Die künstliche Evolution wird vor allem dann verwendet, wenn gewöhnliche Algorithmen in angemessener Zeit keine Lösung finden, da eine diskontinuierliche und/oder nicht differenzierbare Funktion optimiert werden soll und wenn zu viele nicht lineare Parameter verwendet werden.

Ein Gebiet, das sich deshalb besonders für diese naturinspirierten Algorithmen eignet, ist die Robotik. Dort entstand das Teilgebiet der Evolutionären Robotik (vgl. Kapitel 2.1). Dort versucht man z.B. Kontrollsysteme für Roboter mit solchen Algorithmen zu optimieren. Das eignet sich deshalb, weil als Controller oft künstliche neuronale Netze(KNN) verwendet werden (vgl. Kapitel 2.3). Die KNN sind ebenfalls auf Grundlage von Kenntnissen aus der Biologie entstanden. Sie versuchen sich die Eigenschaften von Zellen im Gehirn und im Rückenmark von Säugetieren Eigen zu machen. Sie sind schon bei kleiner Größe sehr schwer per Hand zu programmieren. Mit Evolutionären Algorithmen allerdings lassen sich oft in kürzerer Zeit bessere Ergebnisse verwirklichen (vgl. Kapitel 3.1).

Deshalb will sich die folgende Arbeit diese Algorithmen ebenfalls zu Nutze machen, indem sie die KNN auf diese Weise entwickelt(vgl. Kapitel 2.4.1), bevor sie in der Easy Agent Simulation (EAS, vgl. Kapitel 4.1) getestet werden.

1.4 Gliederung

Die Arbeit ist wie folgt gegliedert:

- Im zweiten Kapitel werden die Grundlagen für die Versuche erklärt.
- Im dritten Kapitel folgt eine Zusammenfassung von verwandten Arbeiten, die sich mit ähnlichen Themen beschäftigt haben.
- Im vierten Kapitel wird die Simulationsumgebung, in der die Versuche durchgeführt werden, und das speziell für diese Versuche entwickelte Programm erklärt.
- Im fünften und sechsten Kapitel werden der genaue Versuchsaufbau und die daraus gewonnen Ergebnisse vorgestellt.
- Im siebten und letzten Kapitel folgt schließlich eine Zusammenfassung und ein Ausblick.

Kapitel 2

Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen für die Arbeit eingeführt. Zunächst wird die Evolutionäre Robotik vorgestellt. Dann folgen zwei spezielle Ansätze, die in der Evolutionären Robotik verwendet werden. Zunächst wird erklärt, was Genetische Algorithmen sind und wie sie mit Evolutionärer Robotik in Beziehung zu setzen sind. Anschließend werden KNN vorgestellt und wie man sie zur Steuerung von Robotern einsetzt. Danach wird der Unterschied zwischen einem Training, das offline, und einem Training, das online stattfindet, dargestellt und schließlich wird erklärt, wie man in einer Simulation Zufallszahlen erzeugt.

2.1 Evolutionäre Robotik

Nach Floreano und Nolfi [FL90] ist die Evolutionäre Robotik eine Technik zur Programmierung von autonomen Robotern. Dabei werden die Roboter als autonome künstliche Organismen angesehen, die in eine Umwelt gesetzt werden und dort ohne den Eingriff von einem Menschen Fähigkeiten entwickeln sollen. Das wird mit aus den Naturwissenschaften abgeleiteten Techniken, wie KNN und Genetischen Algorithmen erreicht, die in den folgenden Abschnitten noch genauer beschrieben werden.

Die Kontrollsysteme der Roboter werden durch künstliche Chromosome repräsentiert. Diese können durch die Gesetze der Genetik verändert und mithilfe von Selektionsmechanismen

ausgewählt werden.

Die Entwicklung der Roboter wird dadurch erreicht, dass man zunächst eine zufällige Population von künstlichen Chromosomen erzeugt. Die Roboter werden dann mit den durch die Chromosomen kodierten Kontrollsystemen (in den folgenden Versuchen KNN) für eine gewisse Zeit in die Umwelt gesetzt. Dort führen sie den Vorgaben ihres Kontrollsystems entsprechend Handlungen aus, die evaluiert werden. Die Evaluation erfolgt mithilfe eines vom Menschen vorher gesetzten Belohnungssystems, bei dem gewisse Aktionen belohnt, andere wiederum bestraft werden. Die Roboter mit dem am besten bewerteten Kontrollsystem geben ihren Code schließlich an die nächste Generation weiter. Der Code wird dann mit Genetischen Operatoren verändert, um eine Entwicklung zu ermöglichen, bevor die nächste Generation wieder in die Umwelt gesetzt wird. Das wird solange durchgeführt, bis keine wesentliche Veränderung mehr eintritt oder im besten Fall das erwünschte Ergebnis erreicht wurde.

2.2 Genetische Algorithmen

Genetische Algorithmen nutzen Prinzipien der natürlichen Evolution. Man erzeugt zunächst eine Population zufälliger künstlicher Chromosome. Die Besten reproduziert man dann selektiv und verändert sie über mehrere Runden (Generationen) zufällig bis man eine optimale Lösung gefunden hat oder ein zuvor festgelegte Abbruchbedingung erfüllt ist.

In der Evolutionären Robotik ist ein Chromosom ein Stringwert, der die Charakteristiken eines Individuums codiert. Im folgenden Szenario werden z.B. die Gewichtungen der Synapsen, die im KNN verwendet werden, mit Real-Werten codiert.

Damit man die Individuen selektieren kann, wird außerdem eine Fitnessfunktion benötigt. Diese beurteilt die Leistung der Individuen in ihrer Umgebung nach jeder Runde. Der Wert, den sie zurückgibt, ist umso höher, je besser sich das jeweilige Individuum verhalten hat.

Der Genetische Algorithmus, der von Goldberg [Gol89] entwickelt wurde, funktioniert folgendermaßen. Man generiert eine zufällige Population von Chromosomen. Dann entschlüsselt man diese, um die jeweiligen Individuen zu erzeugen. Diese werden in die Umwelt gesetzt und nach

einer Generation evaluiert. Nun wendet man drei Genetische Operatoren an, um eine neue Generation zu erhalten. Die drei Genetischen Operatoren sind Selektion, Rekombination und Mutation. Das wird so oft wiederholt, bis man das gewünschte Individuum erhält oder der beste Fitnesswert nicht mehr wächst.

Mithilfe der Selektion erzeugt man eine neue Generation, mit der Rekombination und der Mutation verändert man dann diese, um den Suchraum weiter zu erforschen.

2.2.1 Selektion

Um das Ergebnis zu verbessern, werden nach dem Evaluieren einer Generation Kopien der Chromosome erzeugt, wobei die besseren mit höherer Wahrscheinlichkeit und damit auch in der Regel öfter kopiert werden. Dabei kommen folgende Methoden zum Einsatz.

- Fitnessproportionale Selektion: Die Wahrscheinlichkeit, dass ein Individuum ausgewählt wird, entspricht dem Verhältnis von eigener Fitness zur Summe der Fitnesswerte der gesamten Population, d.h. ein Individuum x_i mit Fitness $f_i = \phi(x_i)$ hat bei einer Populationsgröße von N Individuen die Wahrscheinlichkeit $p_i = \frac{f_i}{\sum_{j=1}^N f_j}$ eine Kopie von sich an die nächste Generation weiter zu geben.

Probleme treten bei dieser Art Selektion auf, wenn entweder alle Individuen sehr ähnliche Fitness haben, da dann die Evolution zu einer zufälligen Suche wird, oder ein Individuum deutlich besser ist als die anderen, da dann die Lösung sehr früh in einem lokalen Optimum konvergiert. Das zweite Problem kann man allerdings mit einer Skalierung, wenn man z.B. eine Obergrenze für die Wahrscheinlichkeit einführt, beheben.

- Rangbasierte Selektion: Man sortiert die Individuen nach Fitnesswerten und berechnet die Wahrscheinlichkeit, mit der ein Genotyp kopiert wird, nicht abhängig vom absoluten Fitnesswert, sondern von der Platzierung in der Liste. So hätte z.B. bei drei Individuen der Beste die Wahrscheinlichkeit $\frac{3}{6}$, der Zweite von $\frac{2}{6}$ und der Dritte von $\frac{1}{6}$.
- gekürztes rangbasiertes Verfahren: Man sortiert wiederum die Individuen nach Fitnesswerten in einer Liste. Man nimmt dann die besten M Individuen und macht von diesen O

Kopien, sodass $M \times O = N$, wobei N die Populationsgröße ist.

- Turnierselektion: Dabei nimmt man zufällig eine der vorher festgelegten Turniergröße entsprechende Anzahl von Individuen aus der Population und vergleicht diese. Das beste Individuum wird in der nächsten Generation wieder verwendet. Das wiederholt man dann N mal.

Außerdem ist es manchmal sinnvoll, in Kombination mit den oben beschriebenen Selektionsmöglichkeiten einfach die besten M Individuen in die nächste Generation zu übernehmen, damit bereits gute gefundene Genotypen nicht verloren gehen. Das wird dann Elitismus genannt.

In der Arbeit wurde die fitnessproportionale Selektion verwendet. Sie wurde deshalb verwendet, weil die anderen Verfahren in diesem speziellen Fall nicht geeignet schienen. Die rangbasierte und die Turnierselektion hätten den guten Agenten eine zu schlechte Möglichkeit geboten, sich durchzusetzen.

2.2.2 Rekombination

Die Rekombination ist nicht immer möglich. So muss der Code auch nach der Rekombination immer noch entschlüsselbar sein. Probleme kann es z.B. geben, wenn man KNN verschiedener Größe kreuzt, da dann Verbindungen zu Neuronen bestehen können, die es im anderen KNN aufgrund einer anderen Größe gar nicht gibt. Deshalb wird die Rekombination auch nicht in dieser Arbeit verwendet. Da sie aber ein wesentliches Element der Evolution darstellt, wird sie im folgenden der Vollständigkeit halber dennoch kurz erklärt:

Es gibt vier Möglichkeiten Individuen zu rekombinieren (vgl. Abb. 2.1):

- One-Point-Crossover: Man bestimmt zufällig eine Stelle im Code, an der die Informationen getauscht werden.
- Multi-Point-Crossover: Man bestimmt mehrere Punkte im Code, an denen die Informationen getauscht werden.
- Uniform-Crossover: Jede zweite Stelle im Code wird getauscht.

1	0	1	1	1	0	0
0	1	1	1	0	0	1

Ursprüngliche Genome

1	0	1	1	0	0	1
0	1	1	1	1	0	0

Genome nach One-Point-Crossover

1	0	1	1	0	0	0
0	1	1	1	1	0	1

Genome nach Two-Point-Crossover

1	1	1	1	1	0	0
0	0	1	1	0	0	1

Genome nach Uniform-Crossover

0.5	0.5	1	1	0.5	0	0.5
-----	-----	---	---	-----	---	-----

Arithmetic-Crossover bei Real-Werten

Abbildung 2.1: Beispiele für Crossover

- Arithmetic-Crossover: Es wird kein Code getauscht. Dafür wird ein neuer Code gebildet, der an jeder Stelle den Mittelwert der zwei Ursprungscodes als Wert hat.

1	0	1	1	1	0	0
1	0	0	1	1	0	0

Genom vor und nach Mutation (Binärwerte)

1.3	2.7	4.2	1.8	0.7	7.2	3.1
1.3	2.7	4.2	1.5	0.7	7.2	3.1

Genom vor und nach Mutation (Realwerte)

Abbildung 2.2: Beispiele für eine Mutation

2.2.3 Mutation

Mutationen sind kleine zufällige Veränderungen im Code, um Varianten bereits existierender Lösungen zu suchen. Der Operator, der die Veränderungen erzeugt, muss so gewählt werden, dass jeder Punkt im Lösungsraum erreicht werden kann. Mutationen eignen sich vor allem, wenn die Lösungen bereits konvergieren, da man so lokale Minima verlassen kann. Die Veränderungen sollten relativ klein sein, damit man die gefundene Lösung nicht komplett verliert. Mutationen werden dadurch erreicht, dass man im Binärcode den Inhalt einer Stelle von 0 auf 1 bzw. umgekehrt tauscht. In dieser Arbeit wird die Mutation bei Realwerten verwendet. In diesem Fall wird eine normalverteilte Zufallsvariable auf den Wert einer Stelle addiert (vgl. Abb. 2.2).

2.3 Künstliche Neuronale Netze

Ein KNN besteht aus Einheiten, Neuronen genannt, die über gewichtete und gerichtete Verbindungen verknüpft sind. Die Verbindungen werden in Anlehnung an die Biologie Synapsen genannt. Nach dem einfachen Neuronen-Modell von McCulloch und Pitts [MP43] ist der Aus-

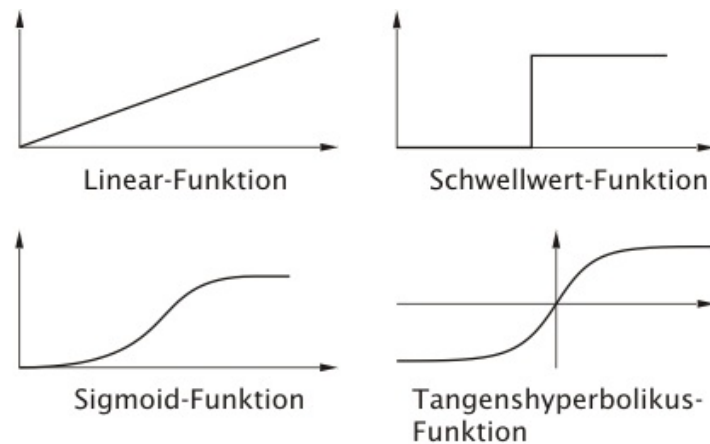


Abbildung 2.3: Aktivierungsfunktionen von Neuronen [Rhe11]

gangswert von einem Neuron eine Funktion der Summe aller gewichteten Eingangswerte.

$$y_i = \phi \left(\sum_j^N w_{ij} x_j \right) \quad (2.1)$$

In manchen Modellen gibt es zusätzlich noch einen Schwellwert, der überschritten werden muss, damit das Neuron überhaupt aktiv wird und einen Wert weitergibt.

Die Funktion ϕ ist im Regelfall eine Treppenfunktion, eine lineare Funktion oder eine Sigmoidfunktion. Anstatt der Sigmoidfunktion wird, wie in dieser Arbeit, auch oft der Tangens Hyperbolicus verwendet, wobei dann die Werte nicht zwischen 0 und 1, sondern zwischen -1 und 1 liegen (vgl. Abb. 2.3).

Jedes Neuron ist entweder ein Input-, ein Output- oder ein verstecktes Neuron und befindet sich dementsprechend im Input-, Output- oder in der versteckten Schicht. Inputneuronen erhalten als Eingangswert einen Sensorwert aus der Umwelt und geben diesen dann in die nächste Schicht vom KNN weiter. Versteckte Neuronen erhalten als Eingangswert entweder den Ausgangswert anderer versteckter Neuronen oder den Ausgangswert von Inputneuronen. Outputneuronen erhalten als Eingangswert ebenfalls den Ausgangswert anderer versteckter Neuronen oder den Ausgangswert von Inputneuronen und geben ihren Ausgangswert direkt an die Aktuatoren in der Umwelt weiter.

Man kann grundsätzlich zwei Arten von KNN unterscheiden. Feedforward-Netze (vgl. Abb. 2.4)

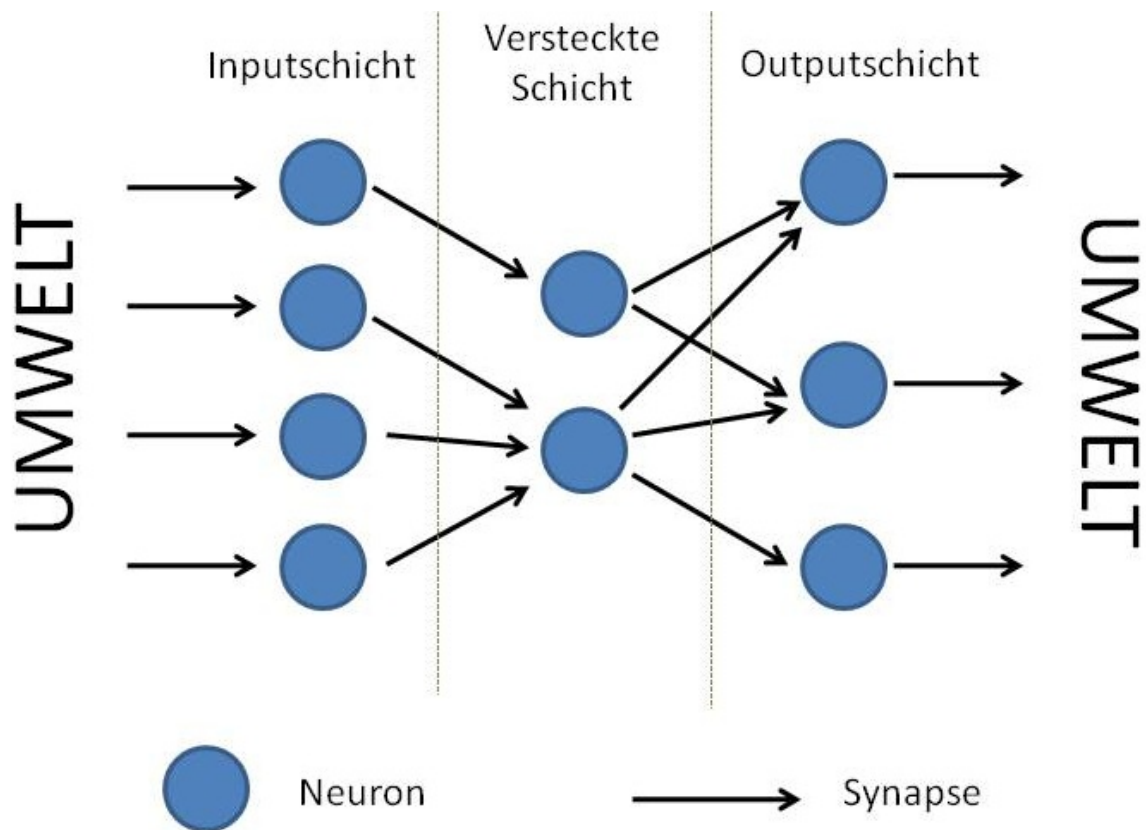


Abbildung 2.4: Beispiel für ein neuronales (Feedforward-)Netz

und rekurrente Netze. Im Gegensatz zu Feedforward-Netzen sind bei rekurrenten Netzen Verbindungen zurück in die vorherige Schicht, innerhalb der gleichen Schicht und sogar zum selben Neuron zurück erlaubt. Dadurch erhalten diese Netze ein sehr komplexes zeitabhängiges Verhalten. Das verschärft zusätzlich ein allgemeines Problem KNN, da es allgemein schon schwer ist das Verhalten der Netze nachzuvollziehen. Damit KNN effizient die gewünschten Ausgabewerte erzeugen, müssen zwei Bedingungen erfüllt sein. Erstens muss das KNN eine Topologie besitzen, die überhaupt das gewünschte Ergebnis zulässt, und zweitens müssen die Verknüpfungen zwischen den Neuronen so sein, dass dieses dann auch erreicht wird. Zum Finden der korrekten Verknüpfungen kann man bei Problemen, bei denen man bereits für einige Fälle sowohl die Eingabe- als auch den damit verbundenen korrekten Ausgabewerte kennt, das KNN mithilfe des Backpropagation-Algorithmus trainieren.

Dabei wird zunächst ein KNN mit zufälligen Verknüpfungen erzeugt. Dann berechnet man den Ausgabewert, den man für einen gegebenen Inputwert mit diesem KNN erreicht, und erhält damit die Abweichung vom korrekten Ausgabewert. Mithilfe dieser Abweichung korrigiert man beginnend bei der Outputschicht bis zurück zur Inputschicht die Gewichtungen. Diesen Vorgang wiederholt man bis das KNN ausreichend genaue Ergebnisse liefert.

Dieser Algorithmus ist in dieser Arbeit allerdings nicht anwendbar, da die gewünschten Ausgabewerte nicht bekannt sind. Außerdem optimiert er nur die Gewichtungen der Verbindungen. Die Topologie bleibt unabhängig davon, ob diese sinnvoll ist oder nicht, erhalten.

Stattdessen werden in dieser Arbeit Elemente der Cascade-Correlation-Architektur von Fahlman und von der NeuroEvolution of Augmenting Topologies verwendet.

2.3.1 Neuro-Evolution

Sowohl die rekurrente Cascade-Correlation-Architektur (rCCA) von Fahlman als auch die NeuroEvolution of Augmenting Topologies (NEAT) von Stanley und Miikkulainen beinhalten nicht nur einen Lernalgorithmus für die Gewichtungen der Synapsen, sondern auch für die Struktur des KNN. Es werden also nicht nur die Verbindungen der Synapsen und deren Gewichtung, sondern auch der Aufbau des KNN als Ganzes, wie z.B. die Anzahl der Neuron, verbessert.

Cascade- Correlation

Bei der rCCA([FL90] und [Fah90]) ist es wie beim Backpropagation-Algorithmus notwendig, dass man Trainingsdaten hat. Man beginnt mit einem KNN minimaler Größe und fügt Schritt für Schritt in der versteckten Schicht neue Neuronen hinzu, sodass ein mehrschichtiges Netzwerk entsteht. Zu Beginn einer Iteration erzeugt man eine Kandidatenmenge von Neuronen. Jedes dieser neuen Neuronen erhält seinen Input von allen Inputneuronen und den schon vorhandenen Neuronen in der versteckten Schicht. Der Output dieser Kandidatenneuronen wird zunächst nicht verwendet. Dann trainiert man die Neuronen mit dem Quickpropalgorithmus [Fah88], einem Verfahren, das sich an das Newton-Verfahren anlehnt, so, dass die Korrelation des Fehlers des Outputs des Neurons mit dem Fehler des aktiven KNN maximal ist. Wenn

die Korrelation nicht weiter wächst, wählt man das beste Kandidatenneuron mit seinen Synapsen. Nun erhält das neue Neuron Synapsen zur Outputschicht und alle Synapsen hin zur Outputschicht werden im gesamten KNN neu trainiert. Diese Iterationen werden so lange wiederholt bis das Abbruchkriterium erfüllt ist. Der Vorteil dieses Algorithmus ist, dass er sehr schnell lernt, er nicht nur die Gewichtungen, sondern auch die Topologie optimiert und dass kein Backpropagation-Algorithmus verwendet werden muss. Da allerdings Trainingsdaten benötigt werden, die für diese Arbeit nicht vorliegen, können nur Teile dieses Algorithmus in diese Arbeit übernommen werden. So wird im dritten Teil der Versuche ebenfalls mit einem KNN minimaler Größe begonnen, das dann im Laufe der Evolution vergrößert wird.

NeuroEvolution of Augmenting Topologies

Bei NEAT [SM02] sind grundsätzlich Synapsen in alle Richtungen zulässig. Also auch z.B. von der Ausgabeschicht zurück in die versteckte Schicht.

Es wird ebenfalls mit einem minimalen KNN gestartet. Dann werden Schritt für Schritt Neuronen und Synapsen hinzugefügt. Es existieren drei Möglichkeiten der Mutation. Es können sich durch die Mutation die Gewichtungen der Synapsen ändern, es können zwischen bereits existierenden Neuronen neue Synapsen eingefügt werden und es können neue Neuronen eingefügt werden. Das Hinzufügen von Neuronen geschieht dadurch, dass eine Synapse aufgeteilt wird und ein Neuron dazwischen gesetzt wird. Die Eingangssynapse, die im neuen Neuron endet, erhält die Gewichtung 1, die Ausgangssynapse die Gewichtung der Vorgängersynapse, die ersetzt wurde. Dadurch wird zunächst die Veränderung durch die Mutation gering gehalten. Die Neuronen und Synapsen werden nummeriert, wodurch man mithilfe von Sequenzen KNN mit ähnlicher Struktur erkennen kann. Dadurch wird es möglich ähnliche KNN zu rekombinieren.

Da eine Mutation oder eine Rekombination möglicherweise zunächst eine Verschlechterung der Fitnesswerte zur Folge hat, werden veränderte KNN für eine gewisse Zeit geschützt. Dadurch sollen neue Lösungen entstehen, mit denen man sich aus lokalen Optima befreien kann.

2.4 Initial Training und Lifelong Adaptation

Das Ziel der Robotik ist, dass sich Roboter in der realen Welt ihnen gestellte Aufgaben lösen können. Da es in der realen Welt ständig Veränderungen gibt oder die Roboter in verschiedenen Umgebungen eingesetzt werden sollen, muss man die Roboter so entwickeln, dass sie sich daran anpassen können. Sie sollten sich deshalb verbessern, wenn die Umgebung stabil ist, und sich anpassen, wenn es eine Änderung in der Umwelt gibt. Nach Walker et al. [WGW06] gibt es dafür zwei Möglichkeiten. Die Training Phase Evolution (TPE) und die Lifelong Adaption By Evolution (LAE).

2.4.1 Training Phase Evolution

Die TPE ist zeitlich beschränkt und findet offline statt. Die Länge wird meistens in Generationen gemessen. Die TPE wird mit Robotern durchgeführt, die entweder gar nicht oder nur sehr schlecht in der Lage sind die ihnen gestellten Probleme zu lösen. Nach Ablauf der TPE ist es möglich, die Roboter mit der LAE weiter lernen zu lassen. Sie findet oft in einer Simulation statt. Diese muss zunächst entwickelt werden und sollte die Realität wahrheitsgetreu abbilden. Außerdem müssen dort die richtigen Parameter für das Training gefunden werden. Arbeitet man mit einer großen Population, kann es lange dauern bis ein zufriedenstellendes Ergebnis erreicht wird. Ist allerdings die Population zu klein, kann es passieren, dass die Diversität nicht ausreichend ist und man nur ein lokales Optimum findet. Es ist also wichtig, dass das Training, vor allem bei einfachen Problemen, nicht zu lange geht, da es sonst wirtschaftlicher wäre eine Lösung per Hand zu programmieren. Außerdem muss das entwickelte Verhalten robust und adaptiv sein.

Es gibt vier Möglichkeiten die TPE durchzuführen:

- Die Evolution und die Anwendung des Kontrollsystems finden auf Robotern statt.
- Die Evolution findet auf Robotern und in einer Simulation statt, die Anwendung dann auf Robotern.

- Die Evolution findet nur in einer Simulation statt, die Anwendung auf Robotern.
- Sowohl die Evolution als auch die Anwendung finden in einer Simulation statt.

In dieser Arbeit wird aufgrund finanzieller und materieller Beschränkungen die vierte Möglichkeit verwendet.

2.4.2 Lifelong Adaption By Evolution

Die LAE ist zeitlich unbegrenzt. Sie läuft parallel dazu, dass sich die Roboter in der Umwelt befinden und ihre Aufgaben erfüllen, ab. Sie findet also online statt. So sollen sich die Roboter permanent anpassen. Sie kann sowohl mit bereits in einer TPE entwickelten Robotern als auch mit Robotern, die noch nicht trainiert wurden, durchgeführt werden. Voraussetzung ist, dass die Roboter ihre eigene Leistung messen und mit einem Fitnesswert beurteilen können. Auch hier benötigt die Entwicklung mit einer größeren Population länger, bietet dafür aber auch eine größere Diversität, die ein schnelles Konvergieren zu einem lokalen Optimum verhindert. Die LAE bietet verschiedene Möglichkeiten. So kann man z.B. die Roboter sich paaren lassen. Dabei tauschen sie zufällig Chromosome und mutieren sie. Das eignet sich vor allem bei Aufgaben, die von mehreren Robotern gelöst werden müssen, da dann die Roboter oft miteinander in Kontakt stehen. Eine andere Möglichkeit ist, dass man so eine Koevolution erreicht. Dass man also, wenn man verschiedene Arten von Robotern hat, die gegenseitig in Konkurrenz stehen, eine wechselseitige Anpassung erreicht. Floreano und Urzelai [FU00] haben gezeigt, dass KNN, die vorher mit der TPE trainiert wurden, sich mithilfe der LAE schnell an dynamische Umwelten anpassen können. Es gibt drei Möglichkeiten die LAE durchzuführen:

- Die Erfahrungen des Roboters in der echten Welt werden zur Evolution genutzt.
- Die Evolution findet sowohl durch Erfahrungen in der realen Welt als auch mithilfe einer Simulation statt.
- Evolution und Anwendung finden in einer Simulation statt.

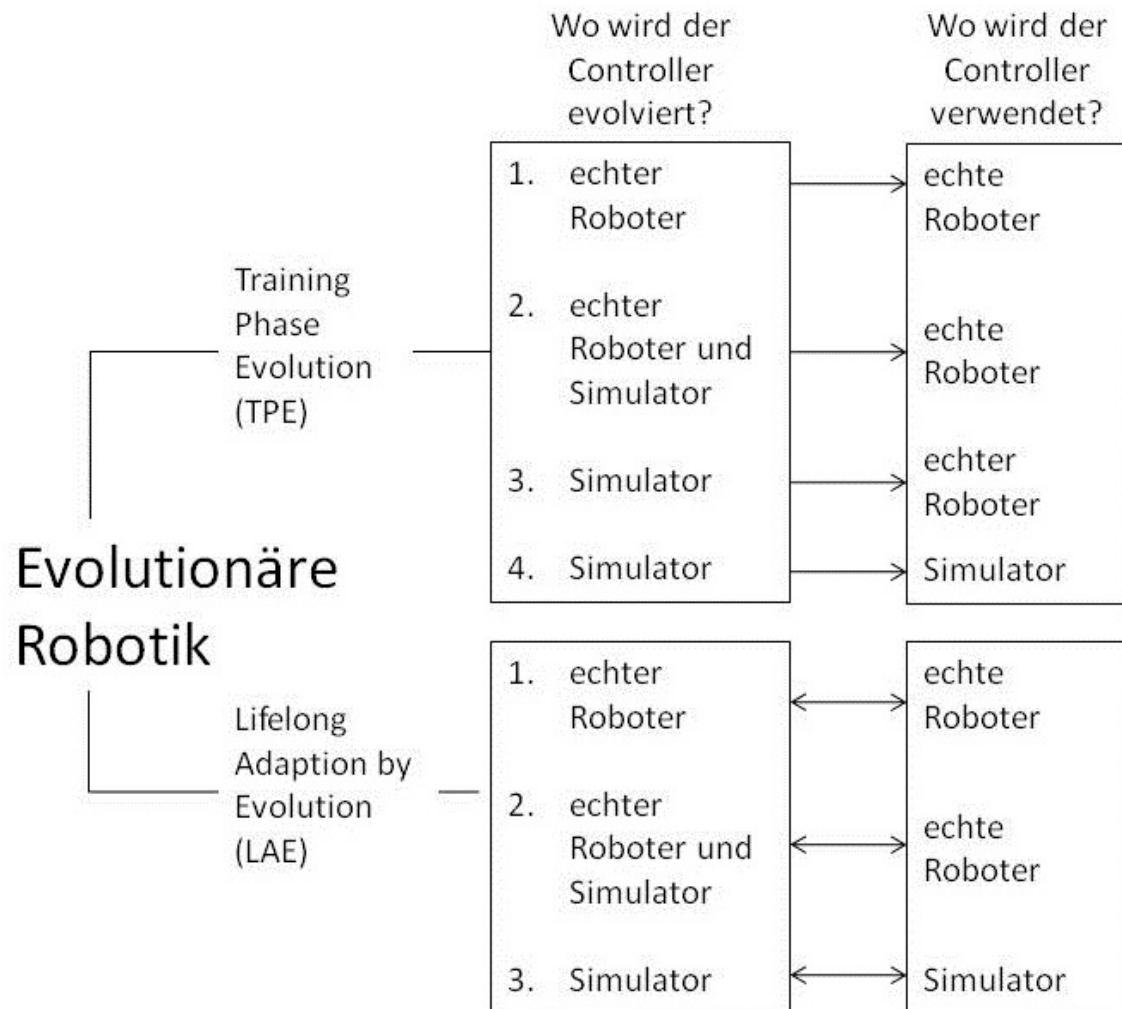


Abbildung 2.5: Möglichkeiten die Training Phase Evolution und die Lifelong Adaption by Evolution durchzuführen. Abbildung nach [WGW06]

Die LAE wird in dieser Arbeit nicht verwendet, da die Agenten keine gemeinsame Aufgabe lösen, wodurch kein regelmäßiger Kontakt zwischen ihnen besteht. Außerdem findet die Simulation in einer stabilen Umwelt statt, sodass sich die Agenten nicht anpassen müssen.

2.5 Pseudozufallszahlen

Die Versuche dieser Arbeit werden mithilfe einer Simulation durchgeführt, die auf der Java-Programmiersprache basiert. Da evolutionäre Prozesse zufällig stattfinden, sind für die Versuche Zufallszahlen notwendig. Diese entstehen aber nicht wie bei einem Würfelwurf wirklich zufällig, sondern werden vom Computer berechnet. In Java entstehen diese mithilfe des Linear-Kongruenz-Algorithmus [KS07].

Der Linear-Kongruenz-Algorithmus ist rekursiv und funktioniert folgendermaßen [Knu97]:

1. Wahl eines Startwertes(Seed) x_0 ,
2. Erzeugen einer Zahlenfolge (x_i) nach der Vorschrift: $x_{i+1} = (a * x_i + c) \bmod m$

Java benutzt dabei folgende Werte für a , c und m

- $a = 25214903917$
- $c = 11$
- $m = 2^{48}$

Diese Formel produziert eine sich immer wiederholende Sequenz. Der Seed, der eine Größe von 48 Bit hat, entscheidet also nur darüber, wo in der Sequenz begonnen wird. Daher kommt auch der Name „Pseudozufallszahlen“. Eine Schwäche dieser Pseudozufallszahlen ist, dass nicht alle Zahlen mit gleicher Wahrscheinlichkeit auftreten. Dafür hat man den Vorteil, dass die Versuche reproduzierbar sind. Da man mit einer festen Sequenz arbeitet, kann man bei einer Wiederholung des Versuchs die gleichen Zahlen erhalten, wenn man mit dem gleichen Startwert beginnt.

Um Pseudozufallszahlen im Bereich von z.B. 0 und 20 zu erhalten, werden die Zahlen, die man durch diesen Algorithmus erhält, in weiteren Schritten durch Verschieben von Bits und weiteren Divisionen auf die gewünschte Größe verkleinert.

Kapitel 3

Verwandte Arbeiten

3.1 Dario Floreano und Francesco Mondada

Als grundlegend im Gebiet der Evolutionären Robotik gelten die Arbeiten von Floreano. In einem Versuch von Floreano und Mondada sollen sich Roboter entwickeln, die sich bewegen und Hindernissen ausweichen [FM98]. Er wurde mit den in der Eidgenössischen Technischen Hochschule Lausanne entwickelten Khepera-Robotern durchgeführt. Diese Roboter verfügen über zwei Räder und acht Infrarotsensoren. Die Motoren der Räder können getrennt angesprochen werden und sowohl Vorwärts- als auch Rückwärtsdrehungen erzeugen. Die Infrarotsensoren können sowohl das Umgebungslicht als auch den Abstand der Roboter zum nächsten Hindernis messen. Sechs davon befinden sich auf der Vorder-, zwei auf der Rückseite des Roboters. Sie können Hindernisse bis zu einem Abstand von ca. 5 cm erkennen. Ihre Werte werden alle 300 ms aktualisiert.

Bis dahin wurden Roboter meist mit einem Braitenberg-Verhalten gesteuert. Bei diesem Konzept werden die Räder direkt mit gewichteten Sensorwerten angesprochen. So kann man z.B., wenn man das linke Rad mit dem linken vorderen Sensor und das rechte Rad mit dem rechten vorderen Sensor anspricht, ein Ausweichverhalten realisieren, da dann jeweils das Rad, das näher am Hindernis ist, mehr beschleunigt wird und der Roboter so dem Hindernis ausweicht. Allerdings müssen die genauen Gewichtungen bekannt sein. Diese unterscheiden sich

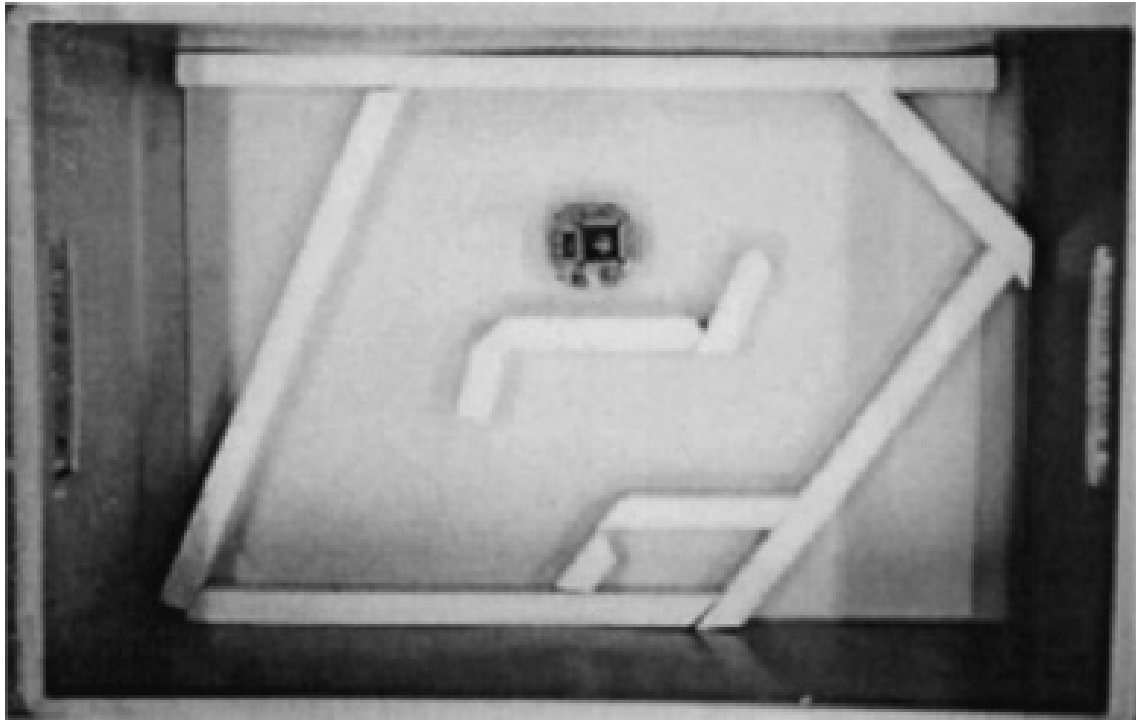


Abbildung 3.1: Spielfeld beim Versuch von Floreano und Mondada [FM98]

aber von Umwelt zu Umwelt und von Roboter zu Roboter. Eine Implementierung per Hand ist dadurch sehr schwer. Als Lösung eignet sich ein evolutionärer Ansatz.

3.1.1 Versuchsaufbau

Bei dem Versuch wurde ein ca. 80 cm mal 50 cm großes Versuchsfeld (vgl. Abb. 3.1) aufgebaut. Der Korridor ist 8 bis 12 cm breit, sodass die Sensoren die meiste Zeit aktiv sind. Das KNN, das den Roboter steuert, besteht nur aus einer Input- und einer Outputschicht. Die Inputschicht besteht aus acht Neuronen, die die acht Werte der Infrarotsensoren als Ausgangswert haben. Diese sind alle mit den beiden Neuronen in der Outputschicht verbunden. Die Outputneuronen bekommen die gewichteten Werte aus der Inputschicht, eine rekurrente Verbindung mit sich selbst und die Werte des anderen Outputneurons als Eingangswert und berechnen dann mithilfe der Sigmoidfunktion den Output. Dieser liegt zwischen -0.5 und 0.5. Dieser Wert

wird direkt an den jeweiligen Motor weitergegeben. Das Vorzeichen gibt dabei die Richtung, der Absolutwert die Geschwindigkeit an. In der Outputschicht gibt es außerdem rekurrente Verbindungen. Die beiden Neuronen sind je mit einem Biasneuron, das immer den Output 1 liefert, über eine Synapse verbunden. Deren Gewichtung wird ebenfalls evolviert. Dadurch wird gewährleistet, dass sich die Roboter auch ohne Sensordaten bewegen können.

Die Fitnessfunktion sieht folgendermaßen aus:

$$\phi = V(1 - \sqrt{\Delta v})(1 - i) \quad (3.1)$$

$$0 \leq V \leq 1$$

$$0 \leq \Delta v \leq 1$$

$$0 \leq i \leq 1$$

d.h. sie besteht aus drei Komponenten. Bei der ersten Komponente (V) werden die Werte der beiden absoluten Werte des Motorenantriebs addiert. Es wird also eine hohe Geschwindigkeit, unabhängig von der Richtung, belohnt.

Bei der zweiten Komponente ($1 - \sqrt{\Delta v}$) wird die Bewegung beider Räder in die gleiche Richtung belohnt. Der höchste Wert wird erreicht, wenn Δv 0 ist. Um Δv zu erhalten, wird 0.5 auf die beiden Motorenwerte addiert. Diese werden dann von einander abgezogen. Schließlich wird der Absolutwert der Differenz genommen.

Die dritte Komponente ($1 - i$) belohnt einen großen Abstand zu Hindernissen. Der Sensor, der am nächsten zu einem Hindernis ist, bestimmt i . i liegt bei 1 bei einem Abstand von 0 cm und fällt dann bis zu einem Minimum von 0 bei einem Abstand von 5 cm oder größer.

Die Population besteht aus 80 Individuen. Zunächst wird das KNN mit kleinen Zufallswerten um den Wert 0 initialisiert. Jedes Individuum wird 80 Sensorzyklen, was etwa 24 s entspricht, getestet und in jedem Zyklus mit der o.g. Fitnessfunktion bewertet. Nach jeder Generation werden die Gewichtungen der Synapsen mit Genetischen Algorithmen verändert. Die durchschnittliche Fitness und die Fitness des Besten werden aufgezeichnet.

3.1.2 Ergebnis und Analyse

Die ersten 100 Generationen wuchs die durchschnittliche Fitness. Dann blieb sie stabil (vgl. Abb. 3.2). Bereits nach 50 Generationen fuhren die Roboter ohne anzustoßen.

Zunächst fuhren die besten Roboter nur gerade und wichen nicht aus. Die Leistung war daher stark abhängig von der Startposition. Sie mussten zunächst eine Balance zwischen den drei Komponenten der Fitnessfunktion finden, da es nicht möglich war alle drei gleichzeitig zu optimieren. Die Balance wurde nach 50 Generationen gefunden. Danach optimierten die Roboter nur noch die erste Komponente, indem sie immer schneller wurden. Obwohl durch die Fitnessfunktion keine Richtung vorgeschrieben wurde, fuhren die besten Individuen in die Richtung, in der sie mehr Sensoren hatten. Die anderen kollidierten öfter mit den Wänden und verschwanden so aus der Population.

Bei Braitenberg-Robotern ist es möglich, dass es zu einem Deadlock kommt. Das passiert z.B. dann, wenn ein Roboter gerade auf eine Wand zufährt. Dann stimmen die linken und die rechten Sensordaten überein und der Roboter kann dadurch nicht ausweichen, da die Netze symmetrisch und geradlinig aufgebaut sind. Das Problem lässt sich zwar mit rekurrenten Verbindungen lösen, ist aber sehr schwer per Hand zu programmieren, ohne andere Verhaltensweisen zu beeinflussen. Die künstliche Evolution hat es aber geschafft.

Die entwickelten KNN waren bei den rekurrenten Verbindungen sehr asymmetrisch. Trotz dieser Asymmetrie kam es zu keinen Problemen beim Navigieren. Deadlocks traten nicht auf.

Als Fazit kann man sagen, dass die Individuen ihr Verhalten den physikalischen Eigenschaften ihrer Sensoren sowie der Umwelt erfolgreich angepasst haben.

3.2 Karl Sims

Die Arbeiten von Karl Sims können ebenfalls als verwandte Arbeiten angesehen werden. Er entwickelte mithilfe von Genetischen Algorithmen virtuelle Kreaturen, die sich in einer physikalischen Umgebung auf verschiedene Weisen fortbewegen können [Sim94].

Die virtuellen Kreaturen bestehen dabei aus vielen durch Gelenke miteinander verbundenen

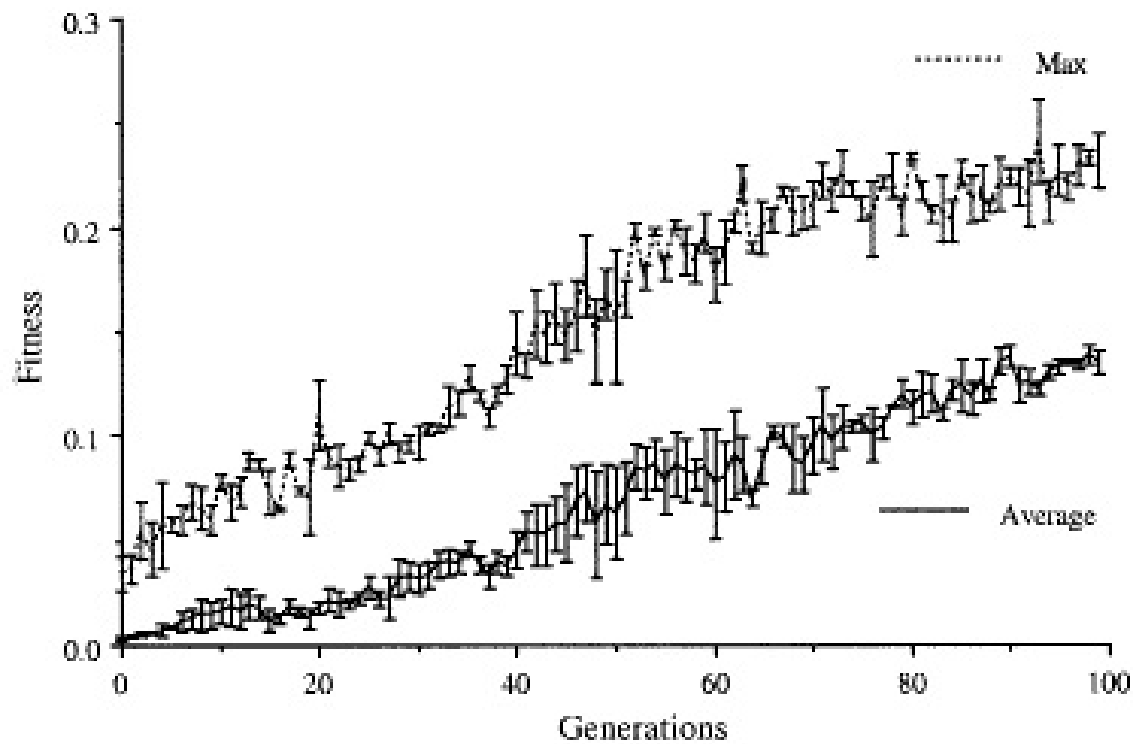


Abbildung 3.2: Fitnesswerte beim Versuch von Floreano und Mondada [FM98]

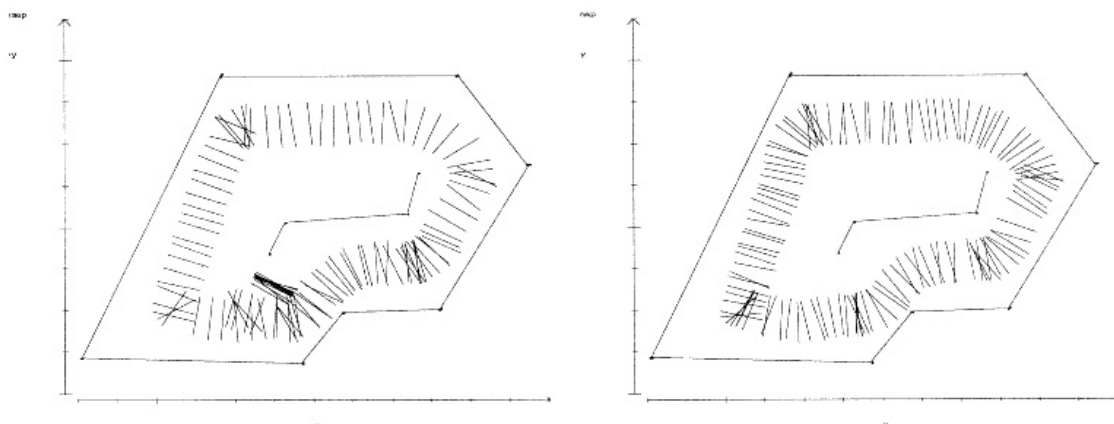


Abbildung 3.3: Trajektorien des besten Roboters der letzten Generation beim Versuch von Floreano und Mondada [FM98]

Boxen. Sie werden ebenfalls durch KNN gesteuert. Als Inputschicht dienen hier die Sensorwerte der verschiedenen Boxen. Diese sind z.B. die Stellung der Gelenke zueinander oder Booleanwerte, die angeben, ob eine Box eine andere Box oder den Boden berührt. Die Neuronen der Outputschicht geben ihre Signale an die Aktuatoren, die die Gelenke bewegen können, weiter. In der versteckten Schicht befinden sich verschiedene Neuronen, die allerdings nicht über die üblicherweise verwendeten Funktionen wie die Treppenfunktion oder die Sigmoidfunktion, sondern über verschiedene Funktionen wie z.B. die Sinus- oder die Exponentialfunktion, aktiviert werden.

Es sollten vier verschiedene Fortbewegungsmöglichkeiten entwickelt werden.

- Schwimmen: Es gibt in der Umgebung keine Gravitation, dafür wird ein Viskositätseffekt aktiviert. In der Fitnessfunktion wird eine hohe Schwimmggeschwindigkeit belohnt.
- Laufen bzw. Bewegung an Land: Gravitation und Reibung am Boden werden aktiviert, die Viskositätseffekt dafür deaktiviert. Eine hohe Geschwindigkeit in horizontaler Richtung wird belohnt.
- Springen: Der Fitnesswert hängt von der maximal erreichten Höhe der tiefsten Box ab.
- Folgen: Es wird eine Lichtquelle aktiviert, die die Kreaturen über Sensoren sehen können. Eine hohe Geschwindigkeit in Richtung der Lichtquelle wird belohnt.

Es wird dann eine zufällige Population von 300 Individuen erzeugt. Die Auswahl für die nächste Generation erfolgt über fitnessproportionale Selektion. Anschließend werden die gefundenen Individuen gekreuzt und mutiert.

3.2.1 Ergebnisse

Bei jedem Versuch konvergierten die Ergebnisse, aber jedes Mal unterschiedlich. Zufriedenstellende Lösungen entwickelten sich nach ca. 50 bis 100 Generationen (vgl. Abb. 3.4).

- Schwimmen: Die Kreaturen bewegten sich wie Wasserschlangen, andere besaßen Flossen oder paddelten.

- Laufen bzw. Bewegung an Land: Es entwickelten sich Kreaturen, die sich ähnlich wie Würmer oder wie Echsen fortbewegten. Andere bewegten sich durch Hüpfen fort.
- Springen: Die Ergebnisse waren nicht sehr vielfältig. Dennoch wurde eine hüpfendes Verhalten erreicht.
- Folgen: Dieses Verhalten wurde sowohl an Land als auch in einer Wassenumgebung untersucht. Die Kreaturen im Wasser entwickelten in diesem Fall teilweise Lenkflossen.

Einfluss auf das Ergebnis kann man als Entwickler bei dieser Evolution nur sehr schwer nehmen. Eine manuelle Auswahl nach jeder Generation wäre zwar möglich, würde aber sehr viel Arbeit bereiten. Als Lösung schlägt Karl Sims eine feste Morphologie der Kreaturen vor, bei der dann nur noch das Gehirn als Steuerungsmechanismus evolviert wird.

3.3 Weitere Arbeiten

3.3.1 Stefano Nolfi und Domenico Parisi

Eine Arbeit von Nolfi und Parisi [NP95] beschäftigt sich mit der Entwicklung von Robotern, die durch ein KNN gesteuert, Gegenstände erkennen, aufnehmen und aus dem Spielfeld tragen sollen. Dafür wurden die Khepera-Roboter zusätzlich mit einem Greifarm ausgestattet. Der Greifarm kann Gegenstände aufnehmen und besitzt zusätzlich einen Sensor, der erkennt, ob sich gerade ein Gegenstand darin befindet. Die Entwicklung fand in einem Simulator statt. Die Sensorwerte, die die Roboter während der Simulation erhalten, wurden vorher in einer realen Umgebung aufgezeichnet, um die Simulation so real wie möglich zu gestalten. Das verwendete KNN bestand aus fünf Sensor- und vier Aktuatorneuronen. Versteckte Neuronen wurden nicht verwendet. Die Sensorneuronen erhielten ihre Werte vom Sensor im Greifarm, von den beiden vorderen Sensoren und einem gemeinsamen Wert der Sensoren links bzw. rechts außen. Die Aktuatorneuronen gaben ihre Werte an die zwei Motoren für die Räder sowie an ein Neuron zum Greifen und eines zum Loslassen der Gegenstände weiter.

Die Fitnessfunktion belohnte folgende Verhaltensweisen:

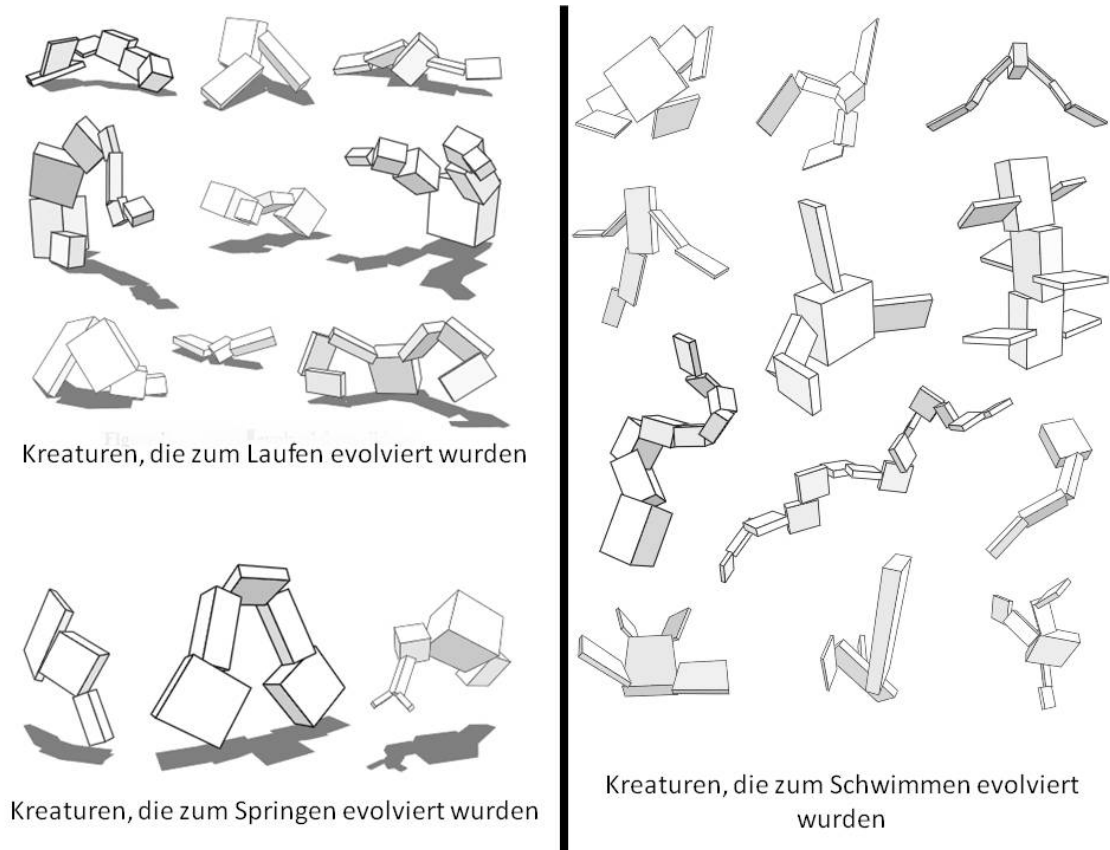


Abbildung 3.4: Entstandene Kreaturen bei den Versuchen von Karl Sims [Sim94]

- der Roboter ist nahe an einem Gegenstand
- der Gegenstand ist frontal vor dem Roboter
- der Roboter versucht den Gegenstand aufzunehmen
- der Roboter hat den Gegenstand im Greifarm
- der Roboter lässt den Gegenstand außerhalb des Spielfeldes los

Es wurden zehn Simulationen mit einer Population von 100 KNN mit zufällig gewichteten Synapsen durchgeführt. Jede Simulation dauerte 300 Generationen lang. In allen zehn Simulationen entwickelten sich KNN, die in der Lage waren, den Roboter in der Simulation wie gewünscht zu steuern. Kein Roboter versuchte die Wand zu greifen, nur in 2% der Fälle schaffte es der Roboter nicht den Gegenstand zu greifen und nur in 10% der Fälle ließ der Roboter den Gegenstand schon im Spielfeld wieder los.

Die besten KNN aus jeder Simulation wurden dann noch auf Robotern in der realen Umgebung getestet. Das Ergebnis war dort etwas schlechter. Der Beste allerdings schaffte es dennoch ohne einen der o.g. Fehler zu machen sechs Gegenstände nacheinander aus dem Spielfeld zu transportieren.

3.3.2 T. Köpsel, A. Noglik und J. Pauli

Eine weitere Arbeit, die verwandte Themen behandelt, ist die von Köpsel, Noglik und Pauli [KNP07]. Sie entwickelten mithilfe von NEAT KNN, die mobile Roboter in unbekannter Umgebung navigieren sollen. Das KNN verbindet dabei die Sensoren des Roboters mit dem Antriebssystem. Der Schwerpunkt lag dabei auf der Entwicklung einer Fitnessfunktion und einer realitätstreuen Simulation für den praktischen Einsatz.

Es wurden drei Fitnessfunktionen getestet. Die erste belohnte zurückgelegte Distanz, gerades Fahren und einen hohen Abstand zum nächsten Hindernis. In der zweiten Fitnessfunktion gab es jeweils Punkte für das Erreichen von Zwischenzielen. Die dritte war dann eine Kombination

aus den ersten beiden. In der Simulation wurden verschiedene Parcours erstellt, die die Roboter zurücklegen mussten. Außerdem wurde ein Mechanismus entwickelt, der das Rauschen der Sensorwerte simulierte. Es wurden dann Versuche mit idealen, teilweise verrauschten und verrauschten Sensorwerten durchgeführt.

Bei den Versuchen mit idealen Sensorwerten entwickelten sich Roboter mit einer höheren Fitness als in den Versuchen mit verrauschten Sensorwerten. Der Übergang in die reale Umgebung funktionierte problemlos, allerdings war das Verhalten in der realen Umgebung von den Robotern, die mit verrauschten Sensorwerten entwickelt wurden, robuster. Außerdem stellte sich heraus, dass Roboter, die von einem KNN mit einfacher Topologie gesteuert wurden, keine schlechteren Ergebnisse lieferten, als die Roboter, die ein KNN mit rekurrenten Synapsen und zusätzlichen Neuronen in der versteckten Schicht besaßen.

Kapitel 4

Implementierung

4.1 Simulationsumgebung

Die Versuche laufen als Simulation ab und werden mithilfe der Easy Agent Simulation (EAS) durchgeführt. Diese ist eine Weiterentwicklung des von Lukas König entwickelten Finite Moore Generators(FMG) [KS08]. Das Framework lässt sich über das Portal sourceforge.net herunterladen [KP11]. In ihm lassen sich beliebige Roboter mit dazugehörigen Sensoren und Aktuatoren entwerfen und simulieren. Als Programmiersprache wird Java verwendet. Im Rahmen der Simulation wird mit einer Physik-Engine gearbeitet. Die Simulation läuft in Zyklen (Ticks) ab, d.h. die Zeit wird diskretisiert. Ein Tick ist die kleinste Zeiteinheit, in der z.B. die Agenten bewegt werden oder auf sie eine Kraft wirkt. 50 Ticks entsprechen etwa einer Sekunde.

EAS bietet außerdem einige Plugins. Mit dem Video-Plugin lässt sich die Simulation visualisieren und mit dem Trajektorien-Plugin kann man sich die Bewegung der Agenten im letzten Zeitraum anzeigen lassen. Es ist weiterhin möglich eigene Plugins zu schreiben. In dieser Arbeit wurde ein Evolutionsplugin und ein Plugin zur Speicherung entworfen.

EAS bietet außerdem die Möglichkeit vor jedem Versuch die jeweiligen Parameter zu ändern, ohne dass man den Programmcode direkt verändern muss. Die in den Versuchen verwendeten Zufallszahlen sind Pseudozufallszahlen (vgl. Kapitel 2.5). Dadurch liefern Versuche, die mit den gleichen Parametern und dem gleichen Seed gestartet werden, das gleiche Ergebnis.

4.2 Umwelt

Die Umwelt, in der die Versuche stattfinden, ist zweidimensional. Sie besteht aus einem Planeten, der sich im Ursprung des Koordinatensystems befindet, und dem Weltall um ihn herum. Vom Mittelpunkt des Planeten aus wirkt in jedem Tick eine Anziehungskraft F_G auf die Agenten. Sie wird folgendermaßen berechnet:

$$F_G = \frac{G * \text{Agentenmasse}}{\text{Abstand}^2}$$

G ist eine Gravitationskonstante, die vom Benutzer über die Parameterwerteingabe gesetzt werden kann. Die *Agentenmasse* wird im Konstruktor des Agenten festgelegt. Der *Abstand* berechnet sich aus der Differenz des Vektors zum Agentenmittelpunkt und des Vektors zum Planetenmittelpunkt. Es ist bei der Formgebung der Agenten darauf zu achten, dass der Mittelpunkt gleichzeitig dessen Schwerpunkt ist, da sonst unrealistische Kräfte wirken würden. Ansonsten wirken keine Kräfte, wie z.B. Reibung, auf die Agenten.

Zur besseren Übersichtlichkeit wird die Grenze des Spielfeldes als roter Kreis im Video- und im Trajektorien-Plug-in angezeigt (vgl. Abb. 4.1).

4.3 Agenten

Die Agenten, die die Raketen darstellen, haben die Form eines spitzwinkligen Dreiecks. Die Spitze soll dabei die Oberseite/Vorderseite symbolisieren. Die Gestalt hat keine Auswirkung auf die physikalischen Kräfte. Sie ist nur für die Berechnung von Kollisionen mit anderen Agenten und für die Auswertung zum Erkennen der Blickrichtung von Bedeutung. Im Konstruktor erhalten sie eine Masse, die für die Berechnung der Gravitationskräfte und der Antriebsbeschleunigungen notwendig ist. Jeder Agent verfügt über fünf Sensoren. Die Werte gelten jeweils vom Agenten aus gesehen.

- Impulssensor vorne: Agentenmasse * Agentengeschwindigkeit in y-Richtung. Der Sensor liefert positive Werte, wenn sich die Rakete nach hinten bewegt, und negative, wenn sie sich nach vorne bewegt (vgl. Abb. 4.2).

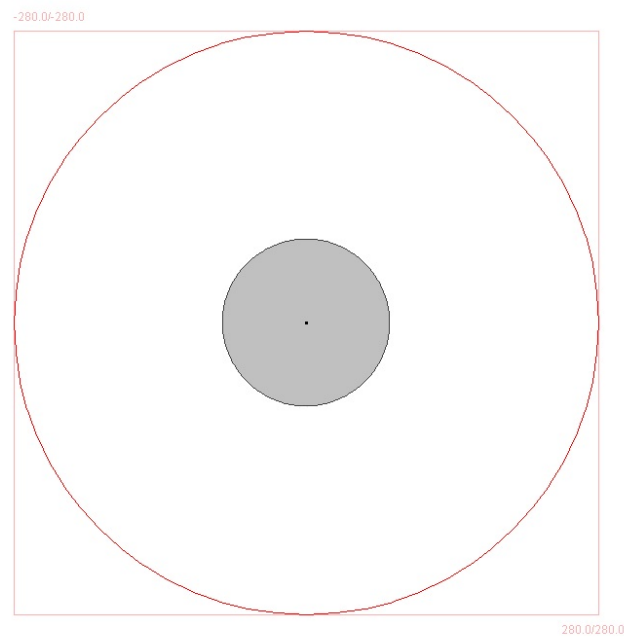


Abbildung 4.1: Umwelt für die Simulation

- Impulssensor seitlich: Agentenmasse * Agentengeschwindigkeit in x-Richtung. Er liefert positive Werte, wenn sich die Rakete nach rechts bewegt, und negative, wenn sie sich nach links bewegt.
- Rotationssensor: Winkelgeschwindigkeit des Agenten. Er liefert positive Werte, wenn sich die Rakete mit dem Uhrzeigersinn, und negative, wenn sie sich gegen den Uhrzeigersinn dreht.
- Kraftsensor vorne: Anziehungskraft in y-Richtung. Der Sensor liefert positive Werte, wenn die Kraft entlang der y-Achse nach hinten wirkt, und negative, wenn die Kraft nach vorne wirkt.
- Kraftsensor seitlich: Anziehungskraft in x-Richtung. Der Sensor liefert positive Werte, wenn die Kraft nach rechts wirkt, und negative, wenn die Kraft nach links wirkt.

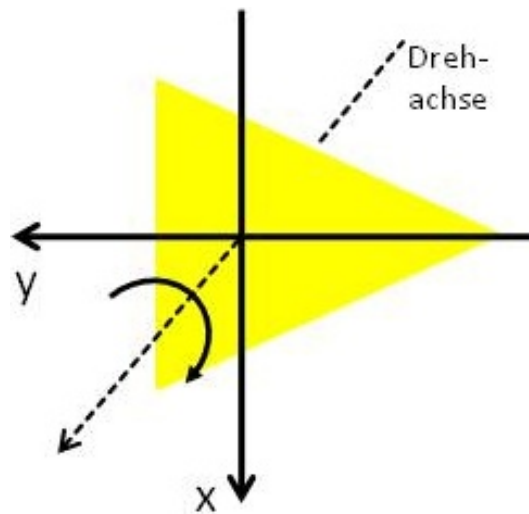


Abbildung 4.2: Koordinatensystem aus Sicht der Rakete. Die Richtung der Achsen gibt an, in welche Richtung der Impuls/die Kraft wirken muss, damit ein positiver Wert gemessen wird. Ein positiver Antrieb wirkt in negative Richtung.

Außerdem hat er zwei Aktuatoren:

- Antrieb: Eine Kraft in y-Richtung wirkt auf den Agenten. Diese kann sowohl nach vorne als auch nach hinten wirken.
- Drehung: Ein Wert wird auf die Winkelgeschwindigkeit des Agenten addiert. Ist der Wert positiv, dreht sich die Rakete im Uhrzeigersinn, ist er negativ, dreht sie sich gegen den Uhrzeigersinn.

4.4 Gehirn

Jeder Agent hat ein Gehirn, das als Controller dient. Es erhält in jedem Tick die Sensordaten und gibt sie an das KNN weiter. Mit den zwei Ausgangswerten des KNN legt es die Stärke der Aktuatoren fest, die ebenfalls in jedem Tick aufgerufen werden. Die Ausgangswerte des KNN, die zwischen -1 und 1 liegen, werden mit am Beginn des Versuches im Parametersatz eingegeben Maximalwerten multipliziert. Wenn dieses Produkt kleiner als 0 ist, wird die Rakete

gebremst bzw. nach links gedreht, bei einem Produkt größer als 0 beschleunigt bzw. nach rechts gedreht.

4.5 Neuronales Netz

Die verwendeten KNN sind eine Weiterentwicklung der KNN, die bereits in den Arbeiten von Benedikt Müller [Mül10] und Christian Nagel [Nag10] verwendet wurden. Sie besitzen zunächst nur fünf Input-, ein Bias- und zwei Outputneuronen. Die Inputneuronen und das Biasneuron bilden die Inputschicht. Die Inputneuronen geben jeweils einen der fünf Sensorwerte an die nächsten Schichten weiter. Das Biasneuron hat ständig den Ausgangswert 1, wodurch auch ohne Sensordaten Aktuatorwerte erzeugt werden können. Die zwei Outputneuronen bilden die Outputschicht. Eines legt die Antriebsstärke, das andere die Stärke der Drehung fest. Ähnlich wie bei NEAT und der Cascade-Correlation-Architektur existieren zunächst keine versteckten Neuronen. Sie können aber im Laufe der Evolution durch Mutationen entstehen, wodurch das KNN erweitert werden kann. Auf die Lernalgorithmen der beiden eben genannten Verfahren wird verzichtet. Stattdessen kommen Genetische Algorithmen zum Einsatz. Zum Verändern der Topologie und zum Verändern der Gewichtungen existieren folgende Genetische Operatoren:

- Neuron einfügen: Es wird ein neues Neuron ohne Synapsen eingefügt. Dieses kann an beliebiger Position zwischen den Eingabe- und Ausgabeneuronen geschehen. Die Eingabe- und Ausgabeneuronen behalten ihre Position am Anfang bzw. am Ende des KNN. Die Indizes der anderen Neuronen werden angepasst.
- Synapse einfügen: Es wird eine Synapse zwischen zwei bereits existierenden Neuronen eingefügt. Diese kann eine beliebige Richtung haben. Als Gewichtung wird dabei ein Zufallswert gewählt, der mit Mittelwert 0 und Standardabweichung 1 normalverteilt ist.
- Neuron löschen: Ein zufällig ausgewähltes Neuron aus der versteckten Schicht wird zusammen mit seinen Synapsen gelöscht. Die Indizes der anderen Neuronen werden angepasst.

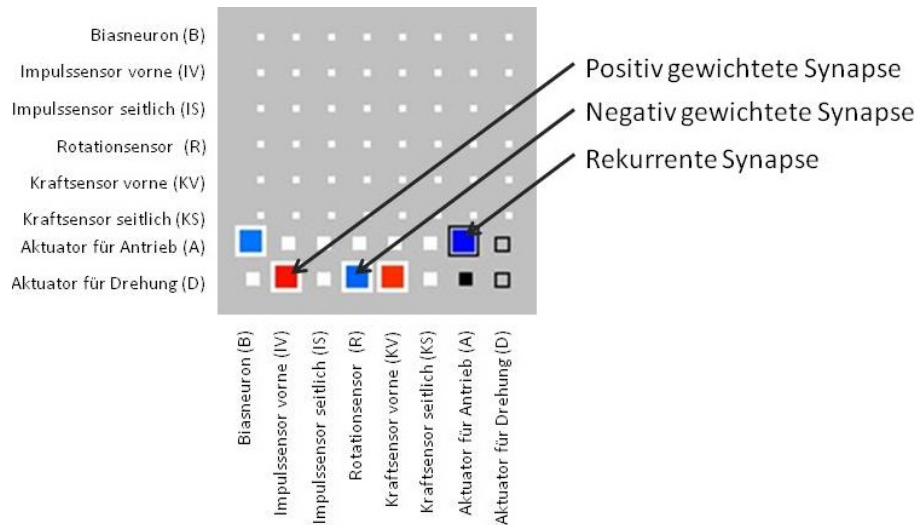


Abbildung 4.3: Grafische Darstellung des verwendeten KNN. Die Richtung der Synapsen ist jeweils von den in den Spalten zu den in den Zeilen genannten Neuronen. Rote Kästchen stellen eine positive Gewichtung, blaue eine negative Gewichtung der Synapse dar.

- Synapse löschen: Eine zufällig ausgewählte Synapse wird entfernt.
- Synapsengewichtung mutieren: Die Gewichtung einer zufällig ausgewählten Synapse wird dadurch verändert, dass ein Zufallswert, der mit Mittelwert 0 und Standardabweichung 1 normalverteilt ist, auf die bestehende Gewichtung addiert wird.

Die Wahrscheinlichkeit, dass bei einer Mutation ein Neuron oder eine Synapse eingefügt wird, ist dabei geringer, als dass ein Neuron oder eine Synapse entfernt wird. Dadurch soll das KNN nach Möglichkeit relativ klein gehalten werden.

Im KNN sind rekurrente Verbindungen erlaubt, d.h. es sind nicht nur Synapsen nach vorne, sondern ebenso nach hinten und zum selben Neuron zurück möglich. Die Neuronen in der versteckten und in der Outputschicht arbeiten mit dem Tangens Hyperbolicus als Aktivierungsfunktion, d.h. sie erzeugen Werte zwischen -1 und 1. Die Ausgabewerte der Inputneuronen können den Sensordaten entsprechend beliebig groß sein. Auf eine Skalierung wurde verzichtet. Diese soll sich über die Gewichtung der Synapsen im Laufe der Evolution entwickeln.

Kapitel 5

Experimente

5.1 Versuchsaufbau

Die für die Physik benötigten Basiseinheiten sind wie folgt definiert:

- Länge: 1 Längeneinheit(LE) $\hat{=}$ $\frac{1}{80}$ vom Radius des Planeten
- Masse: 1 Masseneinheit (ME) $\hat{=}$ Masse eines Agenten
- Zeit: 1 Zeiteinheit(ZE) $\hat{=}$ 1 Tick

Bei allen Versuche wurde eine Population von 100 Agenten erzeugt. Die Simulation dauerte 100000 Ticks. Die Agenten erhielten zunächst Gehirne mit minimalen KNN, die vor dem Start 50 mal mutiert wurden. In den ersten beiden Versuchsreihen wurden nur die Mutationsoperatoren „Synapse einfügen“, „Synapse entfernen“ und „Synapsengewichtung ändern“ aufgerufen. In der dritten Versuchsreihe waren dann alle erlaubt. Die Operatoren „Neuron einfügen“ und „Synapse einfügen“ hatten eine Wahrscheinlichkeit von 0.1. Die Operatoren „Neuron entfernen“, „Synapse entfernen“ und „Synapsengewichtung ändern“ hatten eine Wahrscheinlichkeit von 0.15. Nach der Mutation wurden die Agenten in die Umwelt gesetzt und hatten 20 Ticks Zeit, bevor festgestellt wurde, ob sie noch am Leben waren. Das war deshalb notwendig, weil in einem Versuch die Raketen vom Planeten aus starteten. Der Planet hatte einen Radius von 80 Längeneinheiten(LE). Die Gravitationskonstante betrug $10 \frac{LE^3}{ME * ZE^2}$. Der maximale Antrieb

war $80 \frac{LE * ME}{ZE^2}$ und der maximale Wert für den Aktuator für die Drehung war $20 \frac{LE}{ZE}$. Ein Agent galt als tot, wenn er sich zu nahe am Planet befand, er also entweder nicht startete oder er bereits abgestürzt war, er sich mehr als 200 LE vom Planet entfernt hatte oder seine Winkelgeschwindigkeit größer als $10 \frac{LE}{ZE}$ war. Die zuletzt genannte Bedingung war notwendig, weil sich bei der Entwicklung herausgestellt hatte, dass sich sonst Verhaltensweisen entwickelten, bei denen sich die Rakete immer schneller drehte. Eine zu schnelle Drehung war aber aufgrund der damit entstehenden Fliehkräfte nicht erwünscht.

Der Fitnesswert wurde in jedem Tick ermittelt und gab die Zeit an, seit der der Agent am Leben war. Jeder überlebte Tick gab einen Punkt.

War ein Agent nicht mehr am Leben, wurde er aus der Umwelt entfernt und sein KNN wurde im Zwischenspeicher gespeichert. Waren bereits mehr als 100 KNN im Zwischenspeicher, wurde das schlechteste gespeicherte KNN überschrieben. Für den toten Agenten wurde ein neuer erzeugt. Dieser erhielt nach fitnessproportionaler Selektion ein KNN aus dem Zwischenspeicher, das dann ein Mal mutiert wurde. Alle 1000 Ticks wurden die KNN noch lebender Agenten in den Zwischenspeicher aufgenommen. Dadurch sollten die guten KNN den neu erzeugten Agenten ebenfalls zur Verfügung stehen.

5.2 Speicherung

Alle 100 Runden wurde die aktuelle Durchschnittsfitness in den Endspeicher übernommen. Des Weiteren wurde dort der beste bisher erreichte Fitnesswert zusammen mit statistischen Werten in einer Liste gespeichert. Die statistischen Werte leiteten sich aus den aktuellen KNN aus dem Zwischenspeicher ab und umfassten die durchschnittliche Anzahl an Neuronen, die durchschnittliche Anzahl der Links aller Neuronen addiert und die durchschnittliche Summe der Gewichtungen aller Neuronen addiert sowie die jeweilige Varianz.

Nachdem die maximale Zahl an Ticks erreicht wurde, wurde die letzte Population aus der Umwelt sowie die Liste mit den Fitness- und den statistischen Werten als Textdokument gespeichert (vgl. Abb. 5.1).

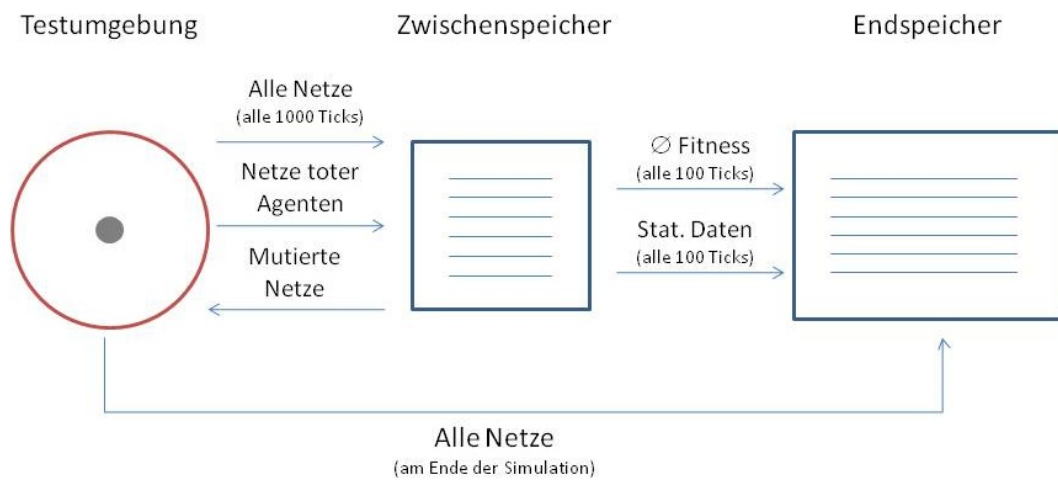


Abbildung 5.1: Ablauf der Speicherung der Ergebnisse

5.3 Verwendete Parameter

Parameter	Wert
Simulationslänge	100000 <i>ZE</i>
Selektion	Fitnessproportional
Zwischenspeichergröße	100
Agenten	100
Speicherung lebender Agenten	Alle 1000 <i>ZE</i>
Anzahl Mutationen vor Simulation	50
Anzahl Mutation bei Vererbung	1
Anzahl versteckte Neuronen bei Beginn	0
Maximalanzahl Neuronen	8, 14

Tabelle 5.1: Übersicht über alle in den Versuchen verwendeten Parameter Teil 1

Parameter	Wert
Wahrscheinlichkeit Neuron einfügen	0.1
Wahrscheinlichkeit Neuron entfernen	0.15
Wahrscheinlichkeit Synapse einfügen	0.1
Wahrscheinlichkeit Synapse entfernen	0.15
Wahrscheinlichkeit Synapse verändern	0.15
Aktivierungsfunktion	\tanh
Maximaler Antrieb	$80 \frac{LE * ME}{ZE^2}$
Maximale Drehung	$20 \frac{LE}{ZE}$
Gravitationskonstante	$10 \frac{LE^3}{ME * ZE^2}$
Mindestlebenszeit	$20 ZE$
Radius Planet	$80 LE$
Maximaler Abstand	$200 LE$
Maximale Drehgeschwindigkeit	$10 \frac{LE}{ZE}$
Abstand beim Start	0.5, 50, 100, 150 LE
Startwinkel	$0, \frac{\pi}{2} LE$

Tabelle 5.2: Übersicht über alle in den Versuchen verwendeten Parameter Teil 2

Kapitel 6

Auswertung

6.1 Erste Versuchsreihe - Start vom Planeten und aus dem Orbit

In dieser Versuchsreihe wurde untersucht, ob sich überhaupt ein Verhalten entwickeln kann, bei dem die Raketen dauerhaft überleben und ob dieses von der Startposition abhängig ist. Die Raketen wurden vom Planeten und aus dem Orbit gestartet. In der Evolution wurden nur minimale KNN, d.h. ohne versteckte Neuronen, bei denen nur die Anzahl der Synapsen und deren Gewichtung optimiert wurde, verwendet. Nach der Evolution wurde analysiert, ob der Abstand vom Planeten für die Ergebnisse eine Rolle spielte. Es wurde bei jedem Versuch das KNN mit dem besten erreichten Fitnesswert betrachtet. Das musste allerdings nicht unbedingt auch das beste KNN sein, da der Fitnesswert eines sehr guten KNN, das erst kurz vor Ende der Simulation entstand, unter Umständen den eines mittelmäßigen KNN, das schon in einer früheren Runde entstanden war, aufgrund der begrenzten Zeit gar nicht mehr einholen konnte.

6.1.1 Start vom Planeten

Bei diesen Versuchen wurden die Raketen mit einem Abstand von 0.5 LE vom Planeten in die Umwelt gesetzt. Sie wurden dabei jedes Mal so platziert, dass die Spitze vom Planeten wegzeigte. Insgesamt wurden auf diese Weise 22 Versuche durchgeführt. Zwei Versuche wurden für ungültig erklärt, weil die Raketen mit dem besten Fitnesswert nur dadurch sich durchsetzten,

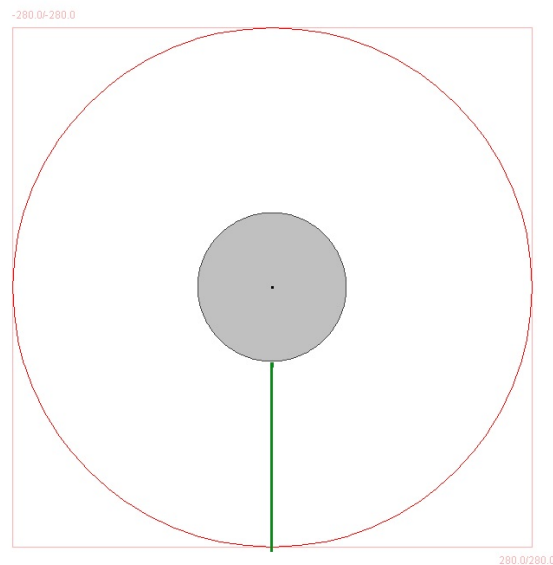


Abbildung 6.1: Flugbahn einer Rakete Kategorie B1 von Tick 0 bis 1500

weil sie vorher mit einer anderen kollidiert waren und so sich ihre Flugbahn positiv verändert hatte.

Nach spätestens 1000 Ticks gab es in jedem Versuch Raketen, die gestartet sind. Auch verbesserte sich in jedem Versuch die durchschnittliche Fitness aller Agenten im Versuch bis zum Schluss. Allerdings waren die Ergebnisse dennoch sehr unterschiedlich. So schaffte es in einem Versuch der beste Agent nur 491 Ticks zu überleben. In einem anderen dafür entwickelte sich ein Agent, der gar nicht abstürzte.

Die entstandenen Verhalten kann man in vier Kategorien unterteilen:

- B1: In 2 Fällen starteten die Raketen und flogen gradeaus aus dem Spielfeld (vgl. Abb. 6.1). Hierbei wurden relativ schlechte Fitnesswerte erzielt. Sie lagen zwischen 1489 und 1668 Punkten. In den KNN gab es verschiedene Verbindungen zwischen Sensoren und Antrieb (vgl. Abb. 6.2). Dadurch starteten die Raketen und flogen gerade.
- B2: In 13 von 20 Fällen begannen die Raketen in Kreisen zu fliegen, die sich wiederum schneckenförmig um den Planeten drehten (vgl. Abb. 6.3). Da die Kreise immer grö-

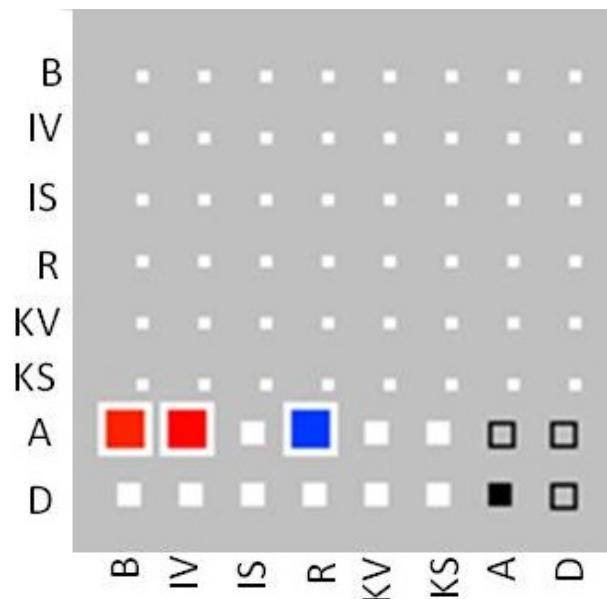


Abbildung 6.2: Beispiel für ein KNN einer Rakete der Kategorie B1. Die Synapsen hin zum Antrieb lassen die Rakete starten und geradeaus fliegen.

ber wurden, stürzten sie irgendwann ab bzw. entfernten sich zu weit. Die erreichten Fitnesswerte lagen zwischen 491 und 95577 Punkten, wobei die Rakete mit einer Fitness von 95577 Punkten noch länger überlebt hätte. Die Fitness konnte nicht weiter steigen, da die Simulation dann endete. Bis auf eine Ausnahme gab es bei allen KNN eine Synapse zwischen Biasneuron und Antrieb und weitere Synapsen hin zum Aktuator für die Drehung, wodurch die kreisförmige Flugbahn entstand (vgl. Abb. 6.4). In einem Fall existierte eine Synapse zwischen Biasneuron und dem Aktuator für die Drehung. Der ständige Antrieb wurde dort durch eine zusätzliche Synapse zwischen Rotationssensor und Antrieb erreicht.

- B3: In 4 Fällen entwickelten sich Raketen, die mehrere Runden um den Planeten flogen (vgl. Abb. 6.5). Sie flogen dann aber entweder aus dem Spielfeld oder stürzten ab. Ihre Flugbahn war jeweils ziemlich gerade, so dass sie keine Kreise in der Flugbahn selbst flogen. Die erreichten Fitnesswerte lagen zwischen 2964 und 23113 Punkten. Die KNN

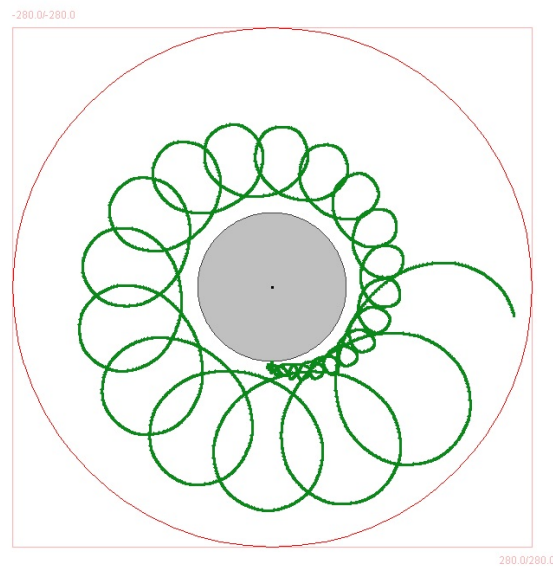


Abbildung 6.3: Flugbahn einer Rakete der Kategorie B2 von Tick 0 bis 10000

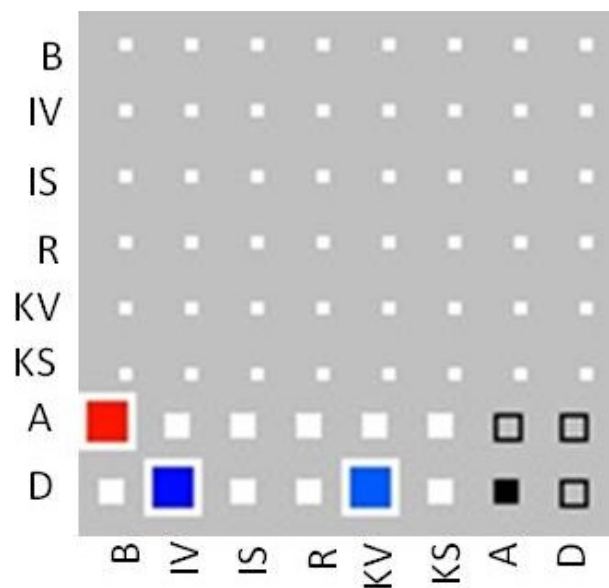


Abbildung 6.4: Beispiel für ein KNN einer Rakete der Kategorie B2. Das Biasneuron sorgt für einen ständigen Antrieb. Durch die Synapsen zwischen Impulssensor vorne bzw. Kraftsensor seitlich und dem Aktuator für die Drehung fliegt die Rakete die kreisförmige Bahn.

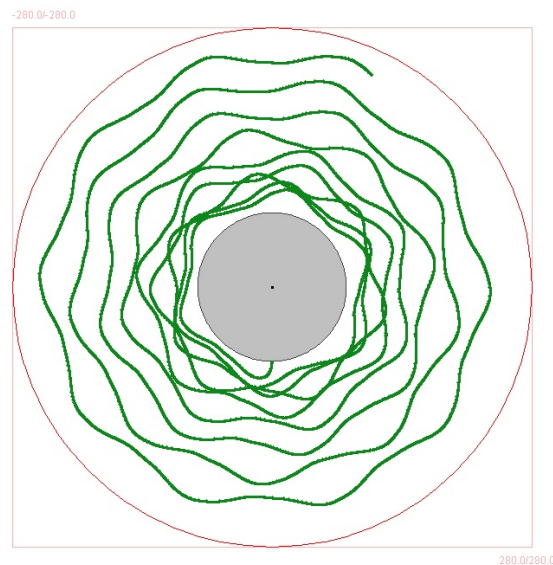


Abbildung 6.5: Flugbahn einer Rakete der Kategorie B3 von Tick 0 bis 20000

waren unterschiedlich. In allen Fällen existierten Synapsen sowohl zum Antrieb als auch zum Aktuator für die Drehung (vgl. Abb. 6.6).

- B4: In einem Fall entwickelte sich eine Rakete, die während der Simulation gar nicht abstürzte. Sie startete und fand dann eine Umlaufbahn um den Planeten (vgl. Abb. 6.7 und Abb. 6.8). Der erreichte Fitnesswert lag bei 39441 Punkten. Dieser sagt allerdings nicht viel aus, da die Rakete am Ende der Simulation noch im Spielfeld war und so weitere Punkte gesammelt hätte. Das KNN besaß fünf Synapsen. Der Antrieb wurde durch das Biasneuron und den Rotationssensor angesprochen, der Aktuator für die Drehung durch das Biasneuron, den Impulssensor vorne und dem Rotationssensor (vgl. Abb. 6.9). Das Biasneuron bremste die Rakete durch die negativ gewichtete Synapse. Allerdings sorgte es auch für eine Linksdrehung, die über den Rotationssensor die Rakete wieder beschleunigte. Der Antrieb und der Aktuator für die Drehung waren also durchgängig aktiv. Die Raketenspitze zeigte immer leicht vom Planeten weg. Entfernte sich die Rakete zu weit, drehte sie sich und der Antrieb wurde stärker aktiv, wodurch sie sich wieder dem

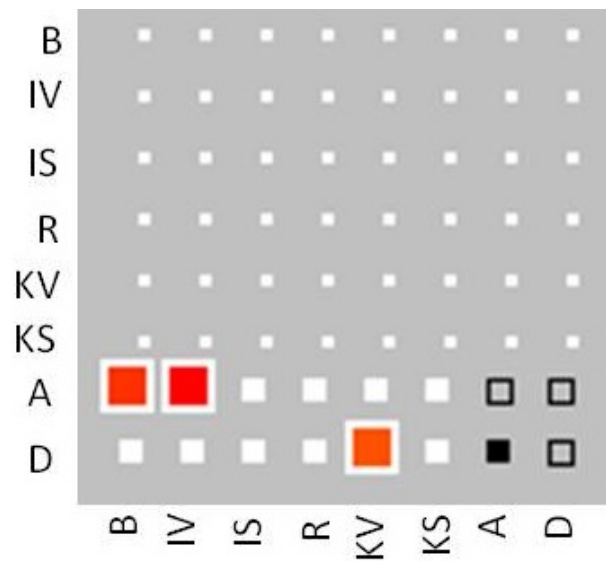


Abbildung 6.6: Beispiel für ein KNN einer Rakete der Kategorie B3. Die Synapsen vom Biasneuron und vom Impulssensor vorne zum Antrieb lassen die Rakete starten und fliegen. Durch die Synapse vom Kraftsensor vorne zum Aktuator für die Drehung fliegt die Rakete im Kreis.

Verhalten	Anzahl	Schlechtestes Ergebnis	Bestes Ergebnis	Durchschnittliches Ergebnis
B1	2	1489	1668	1578.5
B2	13	491	95577*	17037.38
B3	4	2964	23113	11506.75
B4	1	39441*	39441*	39441

Tabelle 6.1: Ergebnisse der Versuche beim Start vom Planeten. *Dieser Wert ist nicht durch Absturz/Entfernen vom Planeten entstanden. Er kam dadurch zustande, dass die Simulation zu Ende war. Der tatsächlich mögliche Fitnesswert ist also höher.

Planeten näherte. Hatte sie sich dann genähert, drehte sie sich wieder so, dass die Spitze leicht wegzeigte, wodurch sie wieder beschleunigte und sich langsam entfernte.

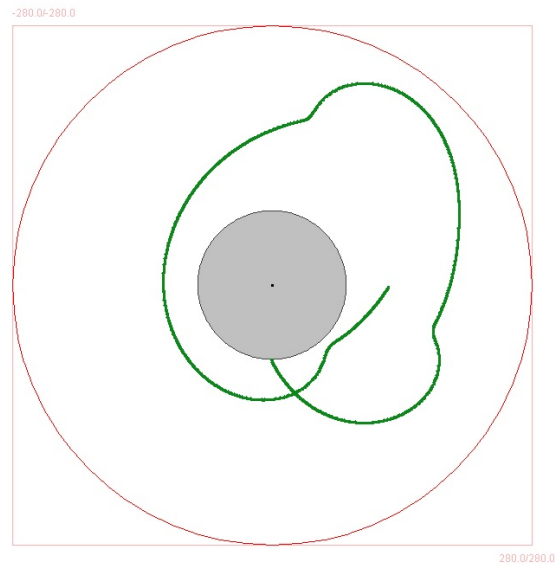


Abbildung 6.7: Start einer Rakete Kategorie B4 von Tick 0 bis 2000

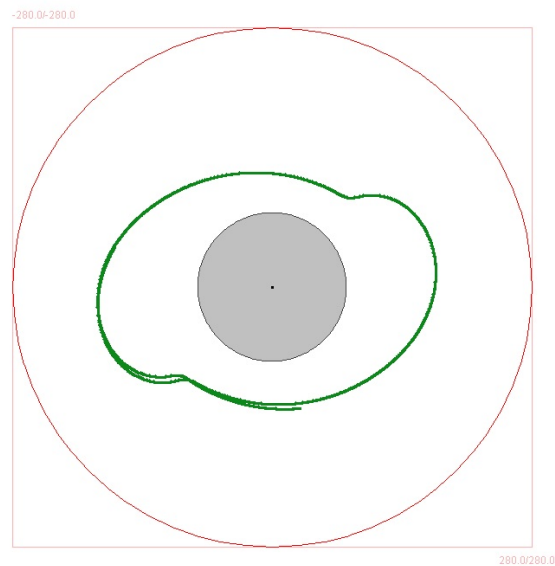


Abbildung 6.8: Flugbahn der Rakete der Kategorie B4 von Tick 6000 bis 8000

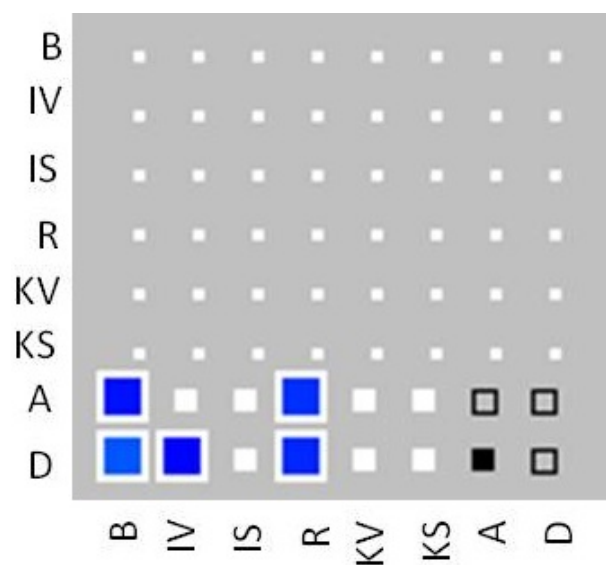


Abbildung 6.9: KNN der Rakete der Kategorie B4. Der Antrieb wird durch das Biasneuron und den Rotationssensor angesprochen, der Aktuator für die Drehung durch das Biasneuron, den Impulssensor vorne und dem Rotationssensor. Der Antrieb und der Aktuator für die Drehung sind also durchgängig aktiv.

6.1.2 Start aus dem Orbit

Bei diesen Versuchen wurden die Raketen mit einem Abstand von 50, 100 oder 150 LE vom Planeten in die Umwelt gesetzt. Sie wurden dabei ebenfalls so platziert, dass ihre Spitze vom Planeten wegzeigte. Insgesamt wurden auf diese Weise je 20 Versuche durchgeführt.

Start mit einem Abstand von 50 LE

Bei den Versuchen mit dem Anfangsabstand von 50 LE entwickelten sich in neun Fällen Raketen, die einen Fitnesswert von 100000 Punkten erreichten.

Die Entwicklungen lassen sich in zwei Kategorien unterteilen:

- L1: Die Raketen mit dem Verhalten L1 blieben alle in der Ausgangsposition, in der die Spitze immer vom Mittelpunkt des Planeten wegzeigte, d.h. es war nur der Antrieb aktiv, der in allen Fällen vom Impulssensor vorne abhängig war. Die Raketen bewegten sich langsam vom Planeten weg bzw. auf ihn zu. Das war davon abhängig, ob die Verknüpfung vom Impulssensor zum Antrieb zu schwach oder zu stark war. Diese Bewegung war allerdings so gering, dass keine Rakete deshalb aus dem Spiel genommen werden musste (vgl. Abb. 6.10). In acht Fällen besaßen die Raketen noch zusätzliche Synapsen, die in dieser Position nicht aktiv waren (vgl. Abb. 6.11). Durch diese Synapsen verliert das Verhalten deutlich an Robustheit. Liefert nämlich ein Sensor einmal einen falschen Wert, weicht also von 0 ab, kommt die Rakete in Bewegung. Das kann z.B. dann, wenn sich die Rakete dreht zu einem Absturz führen. Dieses Verhalten wurde dadurch entdeckt, dass bei Verwendung eines anderen Seeds bei der Auswertung als beim Versuch die Sensorwerte leicht abweichen.
- L2: In zwei Fällen entwickelten sich Raketen, die um den Planeten kreisten. Allerdings entfernten sie sich bei jeder Runde etwas weiter bis sie schließlich das Spielfeld verließen (vgl. Abb. 6.12). Beim Flug zeigt die Spitze der Rakete vom Mittelpunkt des Planeten weg. Das wurde dadurch erreicht, dass eine Synapse den Impulssensor vorne mit dem Antrieb verband, was den Absturz verhinderte. Eine zweite verband den Impulssensor

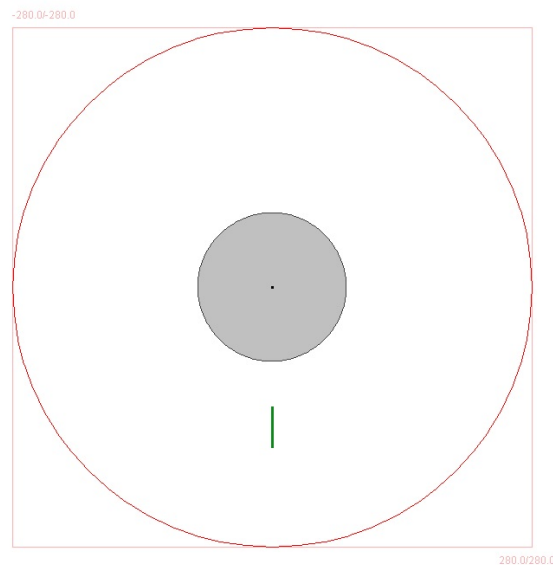


Abbildung 6.10: Flugbahn der Rakete der Kategorie L1 von Tick 0 bis 99999 aus der Starthöhe 50 LE

vorne mit dem Aktuator für die Drehung (vgl. Abb. 6.13). Durch die Drehung blieb die Rakete nicht auf der Stelle und eine kreisförmige Umlaufbahn wurde erreicht. Das Verhalten war also grundsätzlich dem vom L1 sehr ähnlich. Es fand kein wirklich gerichteter Flug statt, sondern es wurde nur durch den Antrieb versucht ein Absturz zu verhindern. Die kreisförmige Flugbahn entstand nur durch geringe Drehungen, die die Rakete daran hinderten auf der Stelle zu bleiben.

Start mit einem Abstand von 100LE

Bei den Versuchen mit dem Abstand von 100 LE gab es 19 Mal am Schluss Raketen, die dauerhaft überlebten. In zehn Fällen entwickelten sich diese bereits in der ersten Runde. Das Verhalten lässt sich in zwei Kategorien einordnen:

- L1: In 19 von 20 Fällen entwickelte sich wieder das Verhalten L1. Die KNN entsprachen denen der Raketen mit Verhalten L1 aus dem Versuch beim Start aus 50 LE Höhe. Zwölf

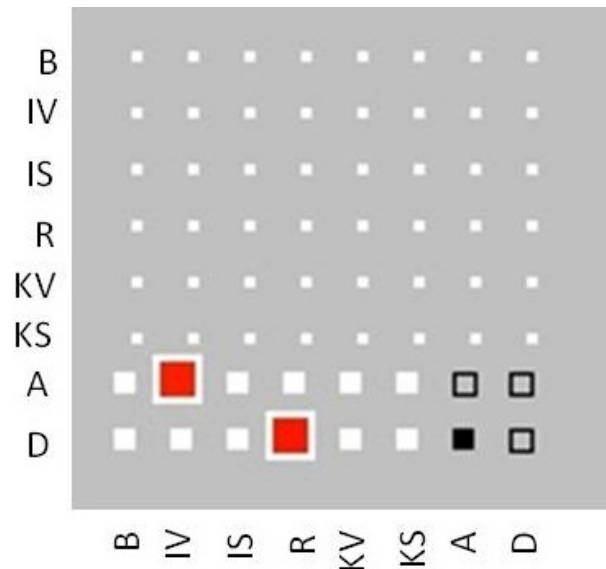
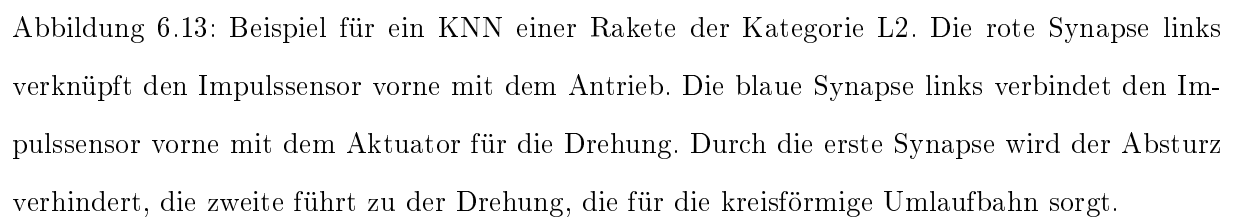


Abbildung 6.11: Beispiel für ein KNN einer Rakete der Kategorie L1. Die Synapse links verknüpft den Impulssensor vorne mit dem Antrieb und ermöglicht es somit, dass die Rakete nicht abstürzt. Sie ist bei allen Raketen der Kategorie L1 vorhanden. Die rechte Synapse ist eine zusätzliche Synapse, die den Rotationssensor mit dem Aktuator für die Drehung verknüpft. In dem getesteten Szenario lieferte der Rotationssensor immer den Wert 0, wodurch die Synapse nicht aktiv war. Sie führt aber in anderen Fällen zu einem instabilem Verhalten. Liefert der Rotationssensor nämlich einmal einen Wert, beginnt sich die Rakete zu drehen, wodurch der Rotationssensor wiederum angesprochen wird, und die Rotation verstärkt. Dadurch dreht sich die Rakete irgendwann zu schnell und muss aus dem Spielfeld genommen werden.

Verhalten	Anzahl	Schlechtestes Ergebnis	Bestes Ergebnis	Anzahl mit Fitness 100000	Durchschnittliches Ergebnis
L1	18	97472*	100000*	9	94036.48
L2	2	97672*	98937*	0	98304.5

Tabelle 6.2: Ergebnisse der Versuche beim Start aus der Höhe 50 LE. *Dieser Wert ist nicht durch Absturz/Entfernen vom Planeten entstanden. Er kam dadurch zustande, dass die Simulation zu Ende war. Der tatsächlich mögliche Fitnesswert ist also höher.



Verhalten	Anzahl	Schlechtestes Ergebnis	Bestes Ergebnis	Anzahl mit Fitness 100000	Durchschnittliches Ergebnis
L1	19	80475	100000*	10	93228.02
L3	1	100000*	100000*	1	100000

Tabelle 6.3: Ergebnisse der Versuche beim Start aus der Höhe 100 LE. *Dieser Wert ist nicht durch Absturz/Entfernen vom Planeten entstanden. Er kam dadurch zustande, dass die Simulation zu Ende war. Der tatsächlich mögliche Fitnesswert ist also höher.

von diesen hatten außer der Synapse zwischen dem Impulssensor vorne und dem Antrieb noch weitere. Dadurch war ihr Verhalten wenig robust. Eine Rakete davon hatte eine zu starke Gewichtung zwischen Impulssensor vorne und Antrieb, wodurch sie sich nach 80475 Ticks zu weit vom Planeten entfernte. Alle anderen überlebten bis zum Ende der Simulation. Zehn davon waren bereits in der ersten Runde entstanden.

- L3: In einem Fall entstand eine Rakete, die sich zunächst dem Planeten näherte, sich dann in eine kreisförmige Umlaufbahn begab und diese dann beibehielt (vgl. Abb. 6.14). Das KNN besaß drei Synapsen (vgl. Abb. 6.15). Die erste verknüpfte den Impulssensor vorne mit dem Antrieb. Die zweite Synapse verband den Kraftsensor seitlich mit dem Aktuator für die Drehung. Die dritte Synapse war eine rekurrente Verbindung mit negativem Gewicht des Aktuatorneurons für die Drehung. Der Drehungsaktuator wurde so gesteuert, dass die Raketenspitze hin zum Mittelpunkt des Planeten zeigte. Dadurch war der Sensorwert für den Impuls vorne zunächst negativ, wodurch sich die Rakete der Anziehung des Planeten widersetzte. Dann wurde der Wert mit der Zeit ungefähr 0 und die Rakete hielt mithilfe der zuvor durch die Annäherung gewonnenen Geschwindigkeit die Umlaufbahn.

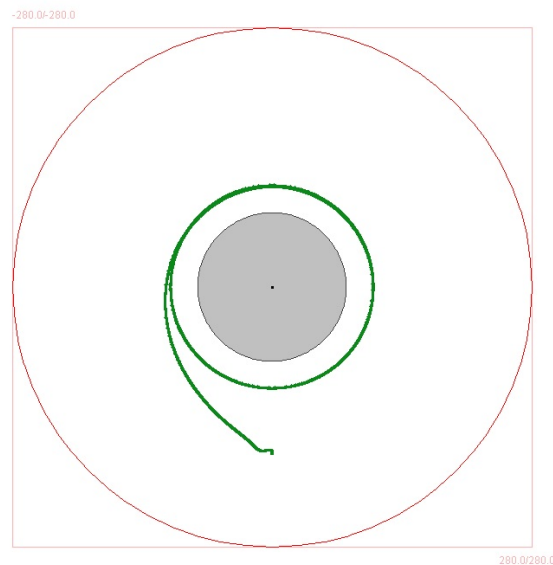


Abbildung 6.14: Flugbahn der Rakete der Kategorie L3 von Tick 0 bis 10000 aus der Starthöhe 100 LE

Start mit einem Abstand von 150LE

Bei einem Abstand von 150 LE gab es 17 Raketen, die einen Fitnesswert von 100000 Punkten erreichten:

- L1: Es entwickelte sich in allen Fällen das Verhalten L1 mit den entsprechenden KNN. 10 waren durch zusätzliche Synapsen weniger robust. Eine davon hatte eine zu schwache Gewichtung der Synapse zwischen Impulssensor vorne und dem Antrieb, wodurch sie einen Fitnesswert von 98799 Punkten erreichte und abstürzte.

Es entwickelte sich also nur in einem Fall, nämlich aus der Starthöhe 100 LE, eine Rakete, die ein komplexeres Verhalten zeigt und nicht nur durch den Antrieb versucht, direkt der Gravitationskraft entgegen zu wirken und so einen Absturz hinauszuzögern.

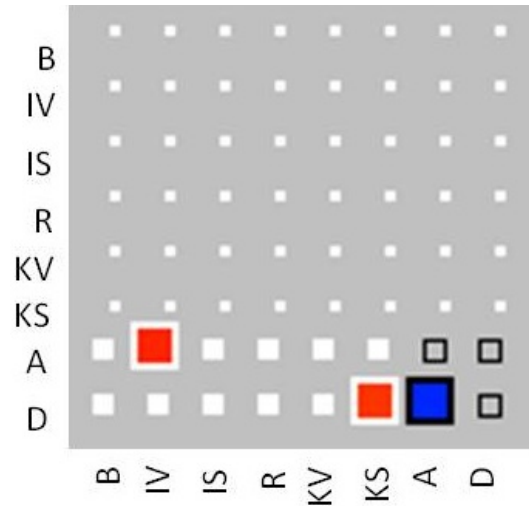


Abbildung 6.15: KNN der Rakete der Kategorie L3. Die Synapse links verknüpft den Impulssensor vorne mit dem Antrieb. Die beiden anderen Synapsen sorgen dafür, dass der Aktuator für die Drehung, so gesteuert wird, dass die Raketenspitze hin zum Mittelpunkt des Planeten zeigt. Der Sensorwert für den Impuls vorne ist zunächst negativ, geht mit der Zeit aber gegen 0. Die Rakete steuert also zunächst leicht gegen den Anziehung und hält dann aber mit der durch die Anziehung gewonnen Geschwindigkeit ihre Umlaufbahn.

Verhalten	Anzahl	Schlechtestes Ergebnis	Bestes Ergebnis	Anzahl mit Fitness 100000	Durchschnittliches Ergebnis
L1	20	98278*	100000*	17	99821.65

Tabelle 6.4: Ergebnisse der Versuche beim Start aus der Höhe 150 LE. *Dieser Wert ist nicht durch Absturz/Entfernen vom Planeten entstanden. Er kam dadurch zustande, dass die Simulation zu Ende war. Der tatsächlich mögliche Fitnesswert ist also höher.

6.1.3 Fazit

Es war also sowohl beim Start vom Boden aus als auch beim Start aus dem Orbit möglich, dass sich KNN entwickelten, die dauerhaft überlebten. Es hat sich aber auch gezeigt, dass die Ergebnisse deutlich besser ausfielen, wenn die Raketen bereits im Orbit starteten. Beim Start aus dem Orbit war bei allen Verhalten die durchschnittliche Fitness höher als 90000 Punkte. Beim Start vom Planeten dagegen erreichte überhaupt nur eine Rakete eine Fitness von über 90000 Punkten. Die Starthöhe von 150 LE scheint die besten Ergebnisse zu liefern. Die entwickelten Verhaltensweisen entstanden aber auch bei den beiden anderen Starthöhen. Aufgrund dieser sehr ähnlichen Ergebnisse kann man also keine Aussage darüber machen, welche Starthöhe beim Start aus dem Orbit am günstigsten ist.

Am interessantesten sind die Verhalten L3 und B4, die sich jeweils nur einmal entwickelten, da beide die deutlich komplexesten der Verhalten sind, die darüber hinaus ein Überleben garantierten.

6.2 Zweite Versuchsreihe - Start aus dem Orbit mit einer Drehung der Raketen um 90 Grad

Da sich in den oben beschriebenen Versuchen beim Start aus dem Orbit sehr oft das simple Verhalten L1 durchgesetzt hatte, wurde eine Veränderung vorgenommen und getestet. Es wurde untersucht, ob bei einer Änderung der Startrichtung andere Verhalten größere Chancen haben sich durchzusetzen. Ziel dieser Versuche war es mehr Raketen, die sich ähnlich wie Raketen vom Typ L3 verhalten, zu entwickeln.

Bei diesen Versuchen wurde die Startrichtung der Raketen um 90 Grad gedreht. Dadurch wurde es deutlich erschwert, dass sich das Verhalten L1 durchsetzt, da nun eine Drehung notwendig war, um mit dem Antrieb die Gravitation direkt auszugleichen. Es wurde ein Abstand von 100 LE gewählt und 20 Versuche durchgeführt. Beispiele für entstandene KNN sind im Anhang zu finden. Bei diesen Versuchen entwickelten sich sechs unterschiedliche Verhalten:

6.2. ZWEITE VERSUCHSREIHE - START AUS DEM ORBIT MIT EINER DREHUNG DER RAKETEN UM

- L3: Es entwickelte sich in zwei Versuchen das Verhalten L3. In einem Fall allerdings entfernte sich die Rakete minimal vom Planeten und hätte daher irgendwann das Spielfeld verlassen. In dem einen Fall, indem sich die Rakete nicht entfernte, hat sich ein KNN entwickelt, das dem aus dem vorherigen Versuch bis auf eine Synapse entsprach. In dem anderen Fall entwickelten sich ein KNN, bei dem der Antrieb durch den seitlichen Kraftsensor angesprochen wurde. Die Drehung erfolgte über den Impulssensor vorne und den Rotationssensor. Wenn die kreisförmige Bahn um den Planeten erreicht waren, hatten sowohl der Kraftsensor seitlich als auch der Impulssensor vorne einen Wert nahe 0. Die Rakete drehte sich dann nur über den Rotationssensor so, dass die Spitze weiterhin zur Mitte zeigte.
- L4: In zwei Fällen entwickelten sich Raketen, die den Planeten umrundeten. Die eine kreisartig (vgl. Abb. 6.16), die andere in Form einer Ellipse. Allerdings wurde die Bahn bei der einen Rakete immer kleiner, wodurch sie irgendwann abstürzen würde, bei der anderen immer größer, wodurch sie irgendwann das Spielfeld verließ. Die Rakete, die in einer Ellipse flog, erreichte einen Fitnesswert von 100000 Punkten. Das KNN bestand bei beiden aus einer Verbindung zwischen dem Kraftsensor seitlich und einer Synapse zwischen Antrieb und Aktuator für die Drehung. Die Rakete, die kreisförmig flog, hatte zusätzlich noch eine Synapse zum Impulssensor vorne.
- L5: In sieben Versuchen entstanden Raketen, die in einem Halbmond flogen (vgl. Abb. 6.17). Allerdings wichen sie immer minimal von dieser Bahn ab. Dadurch stürzten sie irgendwann ab (vgl. Abb. A.1) oder verließen das Spielfeld. Die Fitnesswerte lagen zwischen 44036 Punkten und 100000 Punkten. Im KNN verband bei allen entstandene Raketen dieses Typs eine Synapse den Impulssensor vorne mit dem Antrieb und eine andere den Kraftsensor seitlich mit dem Aktuator für die Drehung. In einem Fall gab es noch eine zusätzliche Synapse, die aber nichts am Verhalten änderte. Die Struktur ähnelte also auch denen der Raketen vom Typ L1, weil hier ebenfalls versucht wurde die Gravitation möglichst direkt mit dem Antrieb auszugleichen.

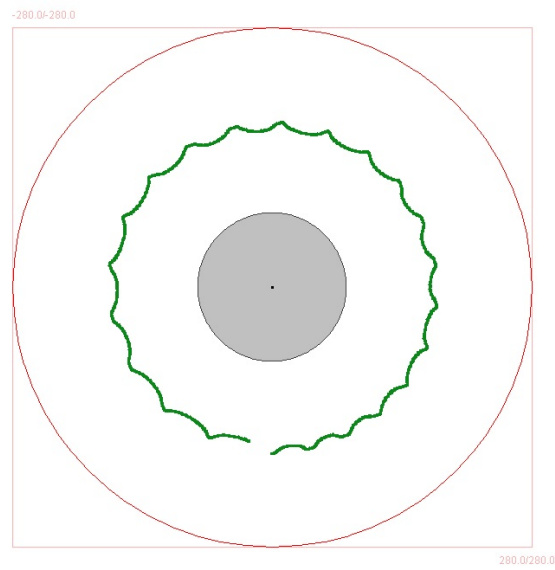


Abbildung 6.16: Flugbahn einer Rakete der Kategorie L4 von Tick 0 bis 15000 aus der Starthöhe 100 LE.

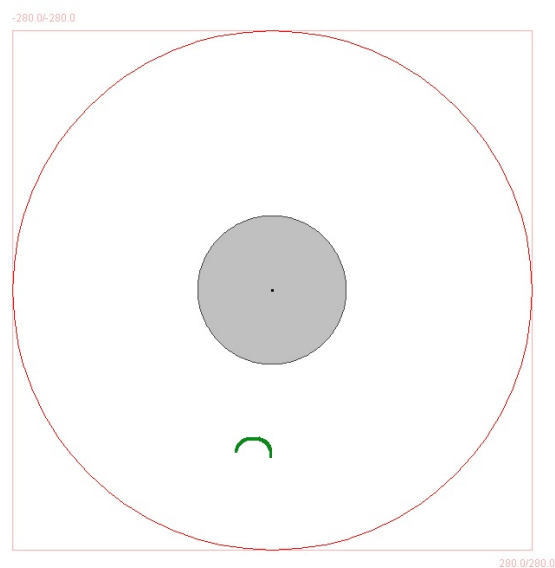


Abbildung 6.17: Flugbahn einer Rakete der Kategorie L5 von Tick 0 bis 1000 aus der Starthöhe 100 LE.

6.2. ZWEITE VERSUCHSREIHE - START AUS DEM ORBIT MIT EINER DREHUNG DER RAKETEN UM

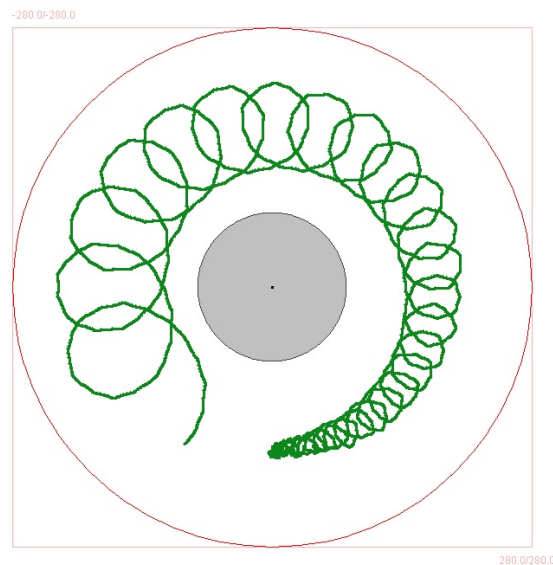


Abbildung 6.18: Flugbahn einer Rakete der Kategorie L6 von Tick 0 bis 12000 aus der Starthöhe 100 LE.

- L6: In sechs Fällen entwickelten sich Raketen, die sich ähnlich wie Raketen vom Typ B2 bewegten. Sie flogen in Kreisen, die sich wiederum schneckenförmig um den Planeten drehten (vgl. Abb. 6.18). Da die Kreise immer größer wurden, stürzten sie irgendwann ab bzw. entfernten sich zu weit. Es wurden Fitnesswerte zwischen 13753 und 99563 Punkten erreicht. Die KNN sahen unterschiedlich aus. In jedem Fall waren aber wieder Antrieb und der Aktuator für die Drehung aktiv, wodurch die kreisförmige Bewegung entstand.
- L7: Es entstanden in zwei Versuchen Raketen, die sich um ihren Mittelpunkt drehten. Dabei bewegten sie sich aber leicht von der Stelle und würden irgendwann abstürzen (vgl. Abb. 6.19). Es waren wieder beide Aktuatoren aktiv. Beide waren am Ende der Simulation noch im Spiel und hatten bis dahin 22337 bzw. 22671 Ticks überlebt.
- L8: In einem Fall entstand eine Rakete, die sich zunächst in einem Dreieck um den Planeten bewegte (vgl. Abb. 6.20), sich dann aber wie eine Rakete vom Typ L6 verhielt. Sie würde irgendwann das Spielfeld verlassen. Am Ende der Simulation hatte sie einen

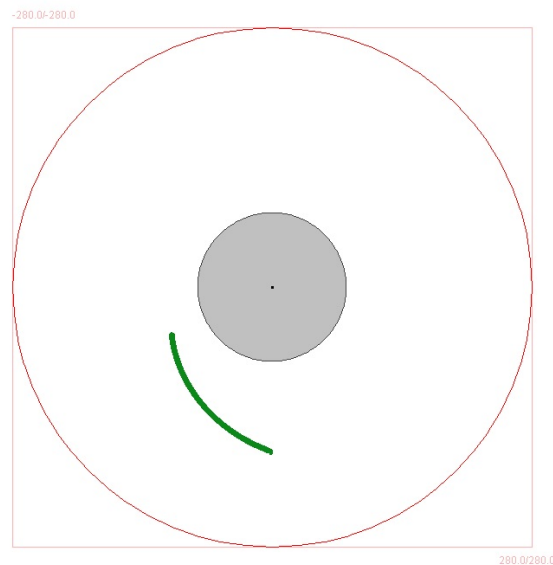


Abbildung 6.19: Flugbahn einer Rakete der Kategorie L7 von Tick 0 bis 50000 aus der Starthöhe 100 LE.

Fitnesswert von 42396 Punkten. Das KNN bestand aus zwei Synapsen. Beide gingen vom Impulssensor vorne aus und führten zum Antrieb und zum Aktuator für die Drehung. Zunächst waren die Sensorwerte noch gering, wodurch ein zwischen den Ecken geradliniger Flug entstand. Irgendwann waren sie dann aber einmal zu hoch, wodurch die Rakete begann kreisförmig zu fliegen. Aus dieser Flugbahn konnte die Rakete nicht zurück in eine stabilere Fluglage gelangen.

6.2.1 Fazit

Es war für die Raketen deutlich schwerer schnell ein Verhalten zu entwickeln, das ein längeres Überleben sicherte. Nur in zwei Fällen entstanden von Beginn an Raketen, die die komplette Simulation überlebten und eine Fitness von 100000 Punkten erreichten. Die Fitnesswerte waren also schlechter als beim Start mit normaler Startrichtung. Immerhin in zwei Fällen wurde das Verhalten L3 erreicht. Ansonsten entstanden neue Verhaltensweisen.

6.2. ZWEITE VERSUCHSREIHE - START AUS DEM ORBIT MIT EINER DREHUNG DER RAKETEN UM

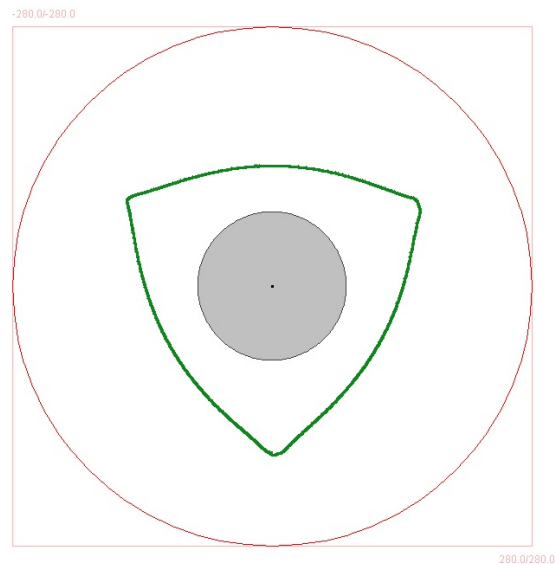


Abbildung 6.20: Flugbahn einer Rakete der Kategorie L8 von Tick 0 bis 5000 aus der Starthöhe 100 LE.

Verhalten	Anzahl	Schlechtestes Ergebnis	Bestes Ergebnis	Durchschnittliches Ergebnis
L3	2	96906*	99563*	98234.5
L4	2	99563*	100000*	99781.5
L5	7	44036	100000*	80975.43
L6	6	13753	99563*	64895.5
L7	2	22337*	22671*	22504
L8	1	42396*	42396*	42396

Tabelle 6.5: Ergebnisse der Versuche beim Start aus der Höhe 150 LE und einer Drehung der Agenten um 90 Grad. *Dieser Wert ist nicht durch Absturz/Entfernen vom Planeten entstanden. Er kam dadurch zustande, dass die Simulation zu Ende war. Der tatsächlich mögliche Fitnesswert ist also höher.

Verhalten	Anzahl	Schlechtestes Ergebnis	Bestes Ergebnis	Anzahl mit Fitness 100000	Durchschnittliches Ergebnis
L1	19	98341	100000*	13	99514.11
L3	1	100000*	100000*	1	100000

Tabelle 6.6: Ergebnisse der Versuche mit Start aus der Höhe 100 LE mit Optimierung der Topologie. *Dieser Wert ist nicht durch Absturz/Entfernen vom Planeten entstanden. Er kam dadurch zustande, dass die Simulation zu Ende war. Der tatsächlich mögliche Fitnesswert ist also höher.

6.3 Dritte Versuchsreihe - Start aus dem Orbit mit Optimierung der Topologie

In diesen Versuchen wurde nun getestet, wie sich die Raketen verhalten, wenn nicht nur die Synapsen und deren Gewichtung, sondern zusätzlich noch die Topologie des Netzes optimiert wird. Dafür wurden die Raketen in eine Höhe von 100 LE gesetzt und zwei verschiedene Startrichtungen jeweils zwanzigmal getestet.

6.3.1 Start mit normaler Ausrichtung

Beim Start mit normaler Ausrichtung entwickelten sich folgende Verhaltensweisen:

- L1: In 19 Fällen entwickelte sich das Verhalten L1.
- L3: In einem Versuch entstand das Verhalten L3.

Damit entstanden die gleichen Verhalten, sogar in der gleichen Anzahl, wie bei den Versuchen aus der gleichen Höhe und der gleichen Startrichtung ohne Optimierung der Topologie. Die durchschnittlichen Ergebnisse beim Verhalten L1 sind mit 99514.11 Punkten etwas besser als die 93228.02 Punkte, die ohne Optimierung der Topologie in der ersten Versuchsreihe beim Start aus 100 LE Höhe entstanden sind.

6.3. DRITTE VERSUCHSREIHE - START AUS DEM ORBIT MIT OPTIMIERUNG DER TOPOLOGIE⁶¹

Analyse der neuronalen Netze

In zwölf Fällen hatten die untersuchten Raketen KNN, die mehr als acht Neuronen besaßen.

Im Schnitt hatten die KNN 9.1 Neuronen. Die Aufteilung sah folgendermaßen aus:

Anzahl Neuronen	Häufigkeit
8	8
9	6
10	4
11	0
12	2
13	0
14	0

In den Fällen, in denen größere KNN entstanden, gab es zwar einige Netze mit Synapsen zwischen den neuen Neuronen in der versteckten Schicht und den anderen Neuronen. Es gab aber nur in einem Fall Synapsen von einem Inputneuron zu einem versteckten Neuron weiter zu einem Outputneuron (vgl. Abb. 6.21). In diesem Fall führten Synapsen vom Impulssensor vorne über das versteckte Neuron zum Antrieb. Sonst hatten die neu entstandenen Neuronen in keinem Versuch Auswirkungen auf das Verhalten der Raketen. Für die entstandenen Verhalten waren die gleichen Synapsen wie bei den Versuchen ohne Optimierung der Topologie zuständig. Die Optimierung der Topologie führte also nicht dazu, dass neue Verhalten entstanden.

6.3.2 Start mit einer Drehung um 90 Grad

Es entwickelten sich folgende Verhaltensweisen:

- L3: In zwei Fällen entwickelte sich das Verhalten L3. In einem Fall entfernte sich die Rakete aber schon während der Simulation deutlich vom Planeten. Sie überlebte dennoch die komplette Simulation.

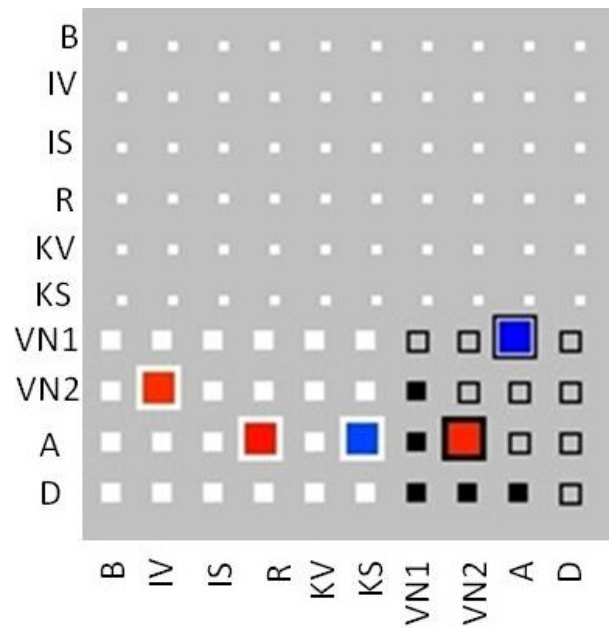


Abbildung 6.21: Beispiel für das KNN der Rakete der Kategorie L1 mit verstecktem Neuron. VN steht für ein verstecktes Neuron. Dieses Netz besitzt als einziges Synapsen vom Impulssensor vorne über das versteckte Neuron zum Antrieb. Hier hat sich also die Topologie so verändert, das sich dies auf die Steuerung auswirkt.

6.3. DRITTE VERSUCHSREIHE - START AUS DEM ORBIT MIT OPTIMIERUNG DER TOPOLOGIE⁶³

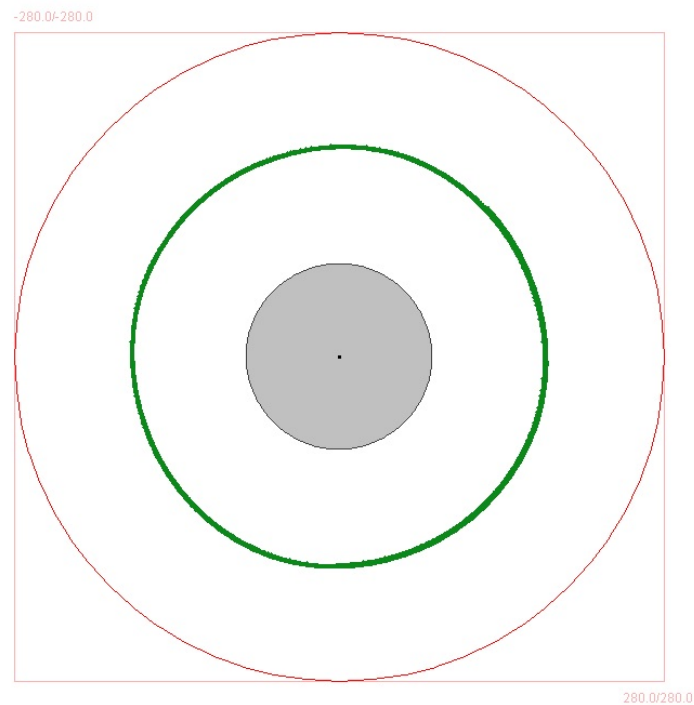


Abbildung 6.22: Flugbahn einer Rakete der Kategorie L4 von Tick 0 bis 10000 aus der Starthöhe 100 LE.

- L4: Das Verhalten L4 entstand ein Mal. Die Flugbahn war deutlich runder als in den Versuchen der zweiten Reihe(vgl. Abb. 6.22). Die Rakete entfernte sich in jeder Runde leicht vom Planeten.
- L5: In fünf Versuchen setzte sich das Verhalten L5 durch.
- L6: Neun Mal entwickelte sich das Verhalten L6.
- L7: In drei Fällen entstand das Verhalten L7.
- L9: In einem Versuch entstand das Verhalten L9. Dabei ähnelte die Flugbahn einer umgedrehten 8 (vgl. Abb. 6.23). Sie näherte sich dabei aber immer weiter dem Planeten. Dennoch überlebte die Rakete die komplette Simulation.

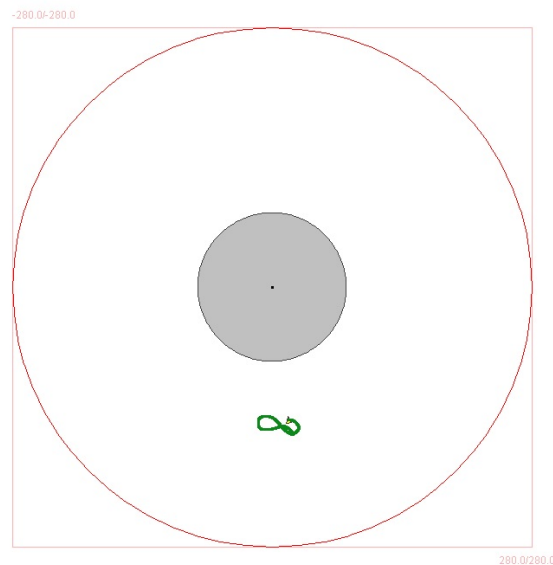


Abbildung 6.23: Flugbahn einer Rakete der Kategorie L9 von Tick 1000 bis 1500 aus der Starthöhe 100 LE.

Die entwickelten Verhalten sind, bis auf eine Ausnahme, die gleichen wie in den Versuchen ohne Optimierung der Topologie. Ebenfalls entstanden die Verhalten L5 und L6 am häufigsten. Es war wiederum möglich eine stabile Umlaufbahn zu finden. Interessant ist das Verhalten L4, da hier eine relativ stabile Umlaufbahn um den Planeten gefunden wurde.

Das durchschnittliche Ergebnis der besten Netze lag mit 76325.94 Punkten über dem Ergebnis von 71981.8 Punkten bei den vergleichbaren Versuchen ohne Optimierung der Topologie in der zweiten Versuchsreihe.

Analyse der neuronalen Netze

Es entwickelten sich in 12 Fällen KNN, die mehr als die Mindestanzahl von acht Neuronen besaßen. Im Schnitt hatten die KNN 9.6 Neuronen. Die Aufteilung sah folgendermaßen aus:

6.3. DRITTE VERSUCHSREIHE - START AUS DEM ORBIT MIT OPTIMIERUNG DER TOPOLOGIE 65

Anzahl Neuronen	Häufigkeit
8	8
9	4
10	3
11	2
12	0
13	2
14	1

Es entstanden zwar wieder Netze mit Synapsen zwischen den ursprünglichen Neuronen und den versteckten Neuronen. Es entwickelten sich aber in keinem Versuch Synapsen von Inputneuronen über versteckte Neuronen zu Outputneuronen. Die neu entstandenen Neuronen hatten also in keinem Versuch Auswirkungen auf das Verhalten der Raketen. Die Verhalten wurden durch die gleichen Synapsen wie in den Versuchen ohne Optimierung der Topologie erzeugt. Die Optimierung der Topologie führte in diesem Szenario zu keiner Änderung der Verhalten.

Verhalten	Anzahl	Schlechtestes Ergebnis	Bestes Ergebnis	Durchschnittliches Ergebnis
L3	2	100000*	100000*	100000
L4	1	77266*	77266*	77266
L5	4	69472*	100000*	85920.75
L6	9	21709	98852*	65252.66
L7	3	20871*	100000*	72765.33
L9	1	100000*	100000*	100000

Tabelle 6.7: Ergebnisse der Versuche beim Start aus der Höhe 150 LE und einer Drehung der Agenten um 90 Grad mit Optimierung der Topologie. *Dieser Wert ist nicht durch Absturz/Entfernen vom Planeten entstanden. Er kam dadurch zustande, dass die Simulation zu Ende war. Der tatsächlich mögliche Fitnesswert ist also höher.

6.3.3 Fazit

Die erreichten Fitnesswerte der Versuche sind etwas besser als die der Versuche ohne Optimierung der Topologie. Da sie aber nur geringfügig abweichen, lässt sich daraus noch nichts schlussfolgern. Die Optimierung der Topologie führte in diesen Szenarien zu keiner wesentlichen Änderung der Verhalten. Die neu entstandenen Neuronen hatten nur in einem von 40 Versuchen überhaupt eine Synapse zu einem Outputneuron. Eine Optimierung der Topologie ist also in diesen Szenarien nicht notwendig, da die gleichen Verhalten auch mit KNN minimaler Größe erreicht werden können.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

In dieser Arbeit wurde untersucht, ob sich mithilfe Evolutionärer Robotik künstliche neuronale Netze entwickeln können, die Raketen so steuern können, dass diese nicht abstürzen und stattdessen eine stabile Umlaufbahn einnehmen. Dafür wurde in der Easy Agent Simulation eine Umwelt mit einem Planeten entworfen. Dieser übt Gravitationskräfte auf die Agenten aus. Es wurde getestet, wie sich die Raketen beim Start vom Planeten und beim Start aus dem Orbit mit zwei verschiedenen Startrichtungen verhalten. Außerdem wurden die Auswirkungen einer Optimierung der Topologie der neuronalen Netze auf die entwickelten Verhaltensweisen untersucht.

Es konnte gezeigt werden, dass die Evolution in allen Fällen erfolgreich verlaufen kann. Die Chance, dass sich Raketen entwickeln, die nicht abstürzen, ist bei einem Start aus dem Orbit größer.

Außerdem zeigte sich, dass die Startrichtung Auswirkungen auf die Verhalten sowie auf die Fitnesswerte hat. Muss sich die Rakete erst drehen, um der Gravitation direkt entgegen zu wirken, verschlechtern sich die Ergebnisse. Eine zusätzliche Evolution der Topologie der neuronalen Netze führte zu keiner Veränderung der Verhalten. Die erreichten Fitnesswerte waren aber etwas besser.

7.2 Ausblick

In zukünftigen Versuchen könnte getestet werden, wie sich unterschiedliche Parameter auf die Verhaltensweisen auswirken. Zudem wären Untersuchungen interessant, die speziell die Evolution der Topologie genauer betrachten. Es wäre noch exakter zu analysieren, ob diese tatsächlich zu einer Verbesserung der Fitnesswerte führt.

Außerdem könnte man die gefundenen Ergebnisse auf ihre Robustheit überprüfen, indem man sie in unterschiedlichen Umwelten testet oder untersucht, wie sich plötzlich auftretende Kräfte auf das Verhalten auswirken.

Ein anderer Aspekt, der analysiert werden könnte, ist die Frage, wie sich Raketen mit mehreren Antrieben verhalten. So ließe sich ein Szenario entwickeln, in dem man mithilfe evolutionärer Algorithmen die Gestalt der Raketen optimiert.

Anhang A

Weitere neuronale Netze

A.1 Start aus dem Orbit aus der Höhe 100 LE

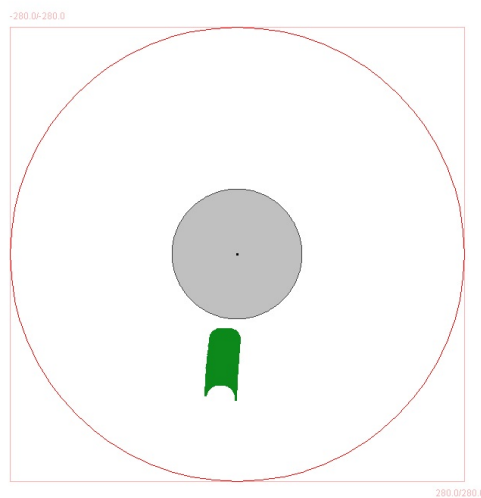


Abbildung A.1: Flugbahn einer Rakete der Kategorie L5 von Tick 0 bis 40000 aus der Starthöhe 100 LE.

A.2 Start aus dem Orbit mit einer Drehung um 90 Grad

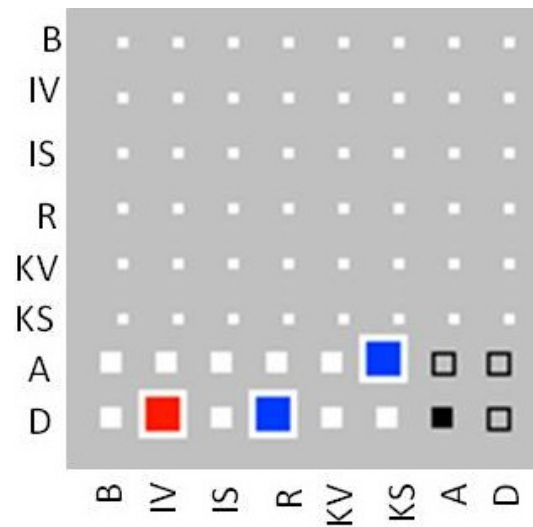


Abbildung A.2: Ein weiteres KNN einer Rakete der Kategorie L3. Der Antrieb wird durch den seitlichen Kraftsensor angesprochen. Die Drehung erfolgt über den Impulssensor vorne und den Rotationssensor.

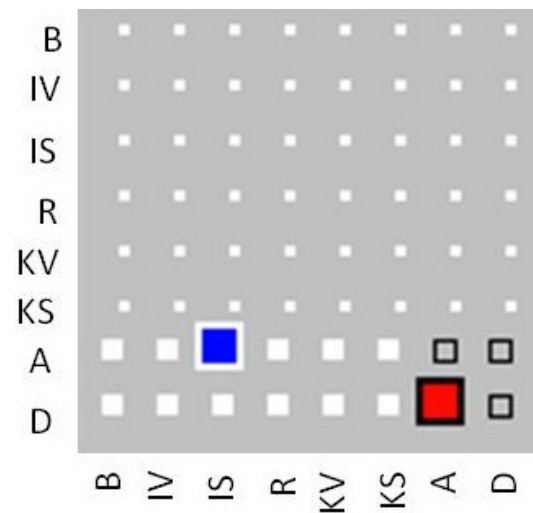


Abbildung A.3: Beispiel für ein KNN einer Rakete der Kategorie L4. Das KNN besteht aus einer Synapse zwischen dem Kraftsensor seitlich und dem Antrieb sowie einer Synapse zwischen Antrieb und Aktuator für die Drehung.

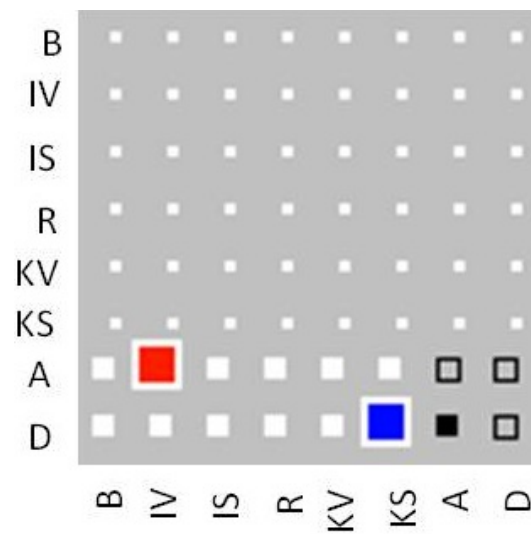


Abbildung A.4: Beispiel für ein KNN einer Rakete der Kategorie L5. Eine Synapsen verbindet den Impulssensor vorne mit dem Antrieb. Die andere den Kraftsensor seitlich mit dem Aktuator für die Drehung.

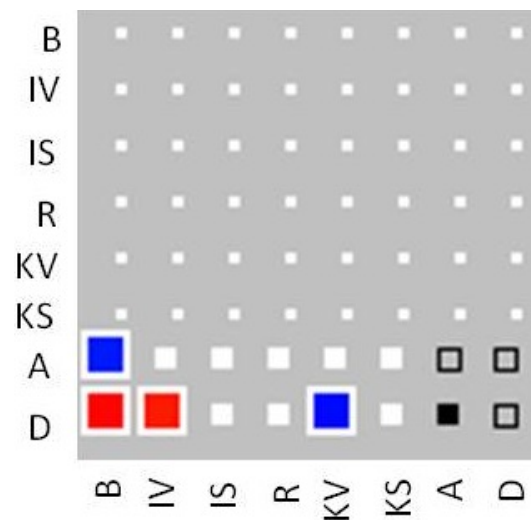


Abbildung A.5: Beispiel für ein KNN einer Rakete der Kategorie L6. Durch die die beiden Synapsen vom Biasneuron zum Antrieb und dem Aktuator kommt die kreisförmige Bewegung zustande.

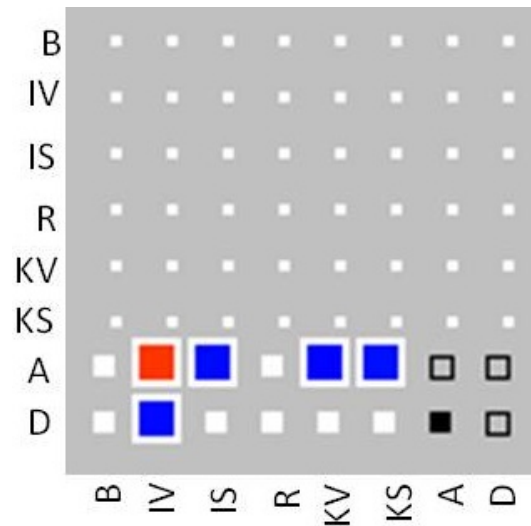


Abbildung A.6: Beispiel für ein KNN einer Rakete der Kategorie L7

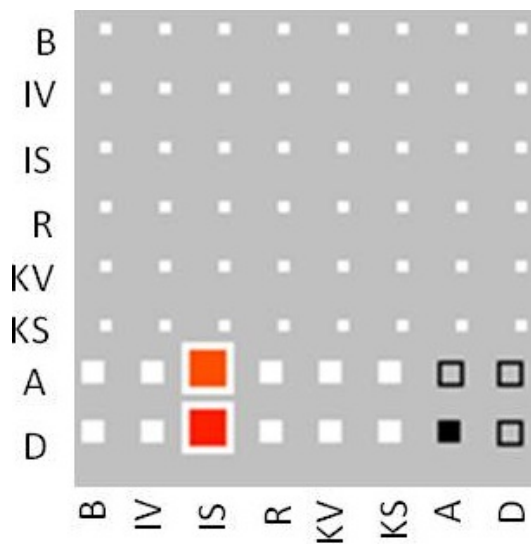


Abbildung A.7: Beispiel für ein KNN einer Rakete der Kategorie L8.

Literaturverzeichnis

- [Dar59] DARWIN, Charles: *On the Origin of Species - A Facsimile of the First Edition*. Cambridge : Harward University Press, 1859
- [Fah88] FAHLMAN, Scott E.: An Empirical Study of Learning Speed in Back-Propagation Networks / School of Computer Science, Carnegie Mellon University Pittsburgh, PA. 1988 (CMU-CS-88-162). – Forschungsbericht
- [Fah90] FAHLMAN, Scott E.: The recurrent cascade-correlation architecture. In: *Proceedings of the 1990 conference on Advances in neural information processing systems 3*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1990 (NIPS-3). – ISBN 1-55860-184-8, 190–196
- [FL90] FAHLMAN, Scott E. ; LEBIERE, Christian: The Cascade-Correlation Learning Architecture / School of Computer Science, Carnegie Mellon University Pittsburgh, PA. 1990 (CMU-CS-90-100). – Forschungsbericht
- [FM98] FLOREANO, Dario ; MONDADA, Francesco: Evolutionary neurocontrollers for autonomous mobile robots. In: *NEURAL NETWORKS 11* (1998), S. 1461–1478
- [FM08] FLOREANO, Dario ; MATTIUSI, Claudio: *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press, 2008. – ISBN 0262062712, 9780262062718
- [FU00] FLOREANO, Dario ; URZELAI, Joseba: Evolutionary robots with on-line self-organization and behavioral fitness. In: *Neural Networks* (2000), S. 431–443

- [Gol89] GOLDBERG, D.E.: *Genetic Algorithms in Search. Optimization, and Machine Learning*. Redwood City, CA : Addison-Wesley, 1989

- [KNP07] KÖPSEL, T. ; NOGLIK, A. ; PAULI, J.: Evolutionäre Algorithmen zur Topologieentwicklung von Neuronalen Netzen für die Roboter-Navigation im praktischen Einsatz. Version: 2007. http://dx.doi.org/10.1007/978-3-540-74764-2_23. In: BRAUER, W. (Hrsg.) ; BERNS, Karsten (Hrsg.) ; LUKSCH, Tobias (Hrsg.): *Autonome Mobile Systeme 2007*. Springer Berlin Heidelberg, 2007 (Informatik aktuell). – ISBN 978–3–540–74764–2, 145-151. – 10.1007/978-3-540-74764-2_23

- [Knu97] KNUTH, Donald E.: *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1997. – ISBN 0201896842

- [KP11] KÖNIG, Lukas ; PATHMAPERUMA, Daniel: *EAS-Framework*. <http://eas-framework.sourceforge.net/>, Mai 2011

- [KS07] KRUEGER, Guido ; STARK, Thomas: *Handbuch der Java-Programmierung*. 6., aktualisierte Ausgabe. Addison-Wesley, 2007

- [KS08] KÖNIG, L. ; SCHMECK, H.: Evolving collision avoidance on autonomous robots. In: HINCHEY, M. (Hrsg.) ; PAGNONI, A. (Hrsg.) ; RAMMIG, F. (Hrsg.) ; SCHMECK, H. (Hrsg.): *Biologically Inspired Collaborative Computing*, 2008, S. 85–94

- [Lam09] LAMARCK, Jean Baptiste Pierre A. d.: *Philosophie zoologique, ou, Exposition des considerations relative a l'histoire naturelle des animaux*. Paris, 1809

- [Men66] MENDEL, Gregor: Versuche über Pflanzenhybriden. In: *Verhandlungen des Naturforschenden Vereines in Brünn IV* (1866), S. 3–47

- [Mül10] MÜLLER, Benedikt: *Neuroevolution in Roboterschwärmen*, Institut für Angewandte Informatik und Formale Beschreibungsverfahren des Karlsruher Instituts für Technologie, Studienarbeit, 2010

- [MP43] McCULLOCH, W. ; PITTS, W.: A logical calculus of the ideas immanent in nervous activity. In: *Bulletin of Mathematical Biophysics* 5 (1943), S. 115–133
- [Nag10] NAGEL, Christian: *Evolution von Kooperation in Roboterschwärmen*, Institut für Angewandte Informatik und Formale Beschreibungsverfahren des Karlsruher Instituts für Technologie, Bachelorarbeit, 2010
- [NP95] NOLFI, Stefano ; PARISI, Domenico: Evolving non-trivial behaviors on real robots: An autonomous robot that picks up objects. Version: 1995. http://dx.doi.org/10.1007/3-540-60437-5_24. In: GORI, Marco (Hrsg.) ; SODA, Giovanni (Hrsg.): *Topics in Artificial Intelligence* Bd. 992. Springer Berlin / Heidelberg, 1995, 243-254. – 10.1007/3-540-60437-5_24
- [Rhe11] RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN - INSTITUT FÜR REGELUNGSTECHNIK (IRT): *Aktivierungsfunktionen von MLP- (oben) und RBF-Netzen (unten)*. <http://www.irt.rwth-aachen.de/29/methoden/identifikation-dynamischer-systeme/knn/>, 2011. – zuletzt abgerufen am 19.07.2011
- [Sim94] SIMS, Karl: Evolving virtual creatures. In: *the 21st annual conference*. New York, New York, USA : ACM Press, 1994. – ISBN 0897916670, 15–22
- [SM02] STANLEY, Kenneth O. ; MIIKKULAINEN, Risto: Evolving Neural Networks through Augmenting Topologies. In: *Evolutionary Computation* 10 (2002), Nr. 2, 99-127. <http://dx.doi.org/10.1162/106365602320169811>. – DOI 10.1162/106365602320169811
- [Wei07] WEICKER, Karsten: *Evolutionäre Algorithmen*. 2. Auflage. Leipzig : B.G. Teubner Verlag / GWV Fachverlage GmbH, 2007 (Leitfäden der Informatik)
- [WGW06] WALKER, Joanne H. ; GARRETT, Simon M. ; WILSON, Myra S.: The balance between initial training and lifelong adaptation in evolving robot controllers. In: *IE-EE transactions on systems man and cybernetics Part B Cybernetics a publicati-*

on of the IEEE Systems Man and Cybernetics Society 36 (2006), Nr. 2, 423-432.

<http://www.ncbi.nlm.nih.gov/pubmed/16602601>

Erklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, die wörtlich oder inhaltlich übernommen Stellen als solche kenntlich gemacht zu haben und die Satzung des Karlsruher Instituts für Technologie (KIT) zur Sicherung guter wissenschaftlicher Praxis beachtet zu haben.

Karlsruhe, den 01.09.2011, Dominik Colling